

MetaPlab 1.1 User Guide

Vincenzo Manca Alberto Castellini Giuditta Franco
Luca Marchetti Roberto Pagliarini

Verona University, Computer Science Department,
Strada Le Grazie 15, 37134 Verona, Italy.

{vincenzo.manca, alberto.castellini, giuditta.franco,
luca.marchetti, roberto.pagliarini}@univr.it

February 26, 2009

Abstract

Metabolic P systems, shortly MP systems, are a special class of deterministic P systems, introduced for modeling biological metabolism. *MetaPlab* is a software written in Java which provides a plug-in based architecture of tools for *i)* defining MP systems, *ii)* computing their dynamics, *iii)* inferring MP systems having the same behavior of an observed metabolic system. In this document we will present a step-by-step guide of the main functionalities of MetaPlab.

1 A quick introduction to MP systems and MP graphs

Metabolic P systems [2, 3, 4, 7, 8, 9, 11, 12, 13, 16, 17, 18, 21, 22, 19, 20, 23, 24], MP systems for short, are a special class of P systems introduced to model quantitative aspects of biological systems, while avoiding the use of complex systems of differential equations. Differently from classical P systems, typically based on non-deterministic evolution strategies, MP systems dynamics is computed by means of a discrete deterministic evolution strategy called Equational Metabolic Algorithm (EMA).

Using MP systems, it has been possible to provide models of several famous biochemical processes, among them: the Belousov-Zhabotinsky reaction (Brusselator) [3, 4], the Lotka-Volterra dynamics [25, 3, 4], a Susceptible-Infected-Recovered epidemic [3], the Leukocyte Selective Recruitment in the immune response [14, 3], the Protein Kinase C Activation [4], Mitotic Cycles [23], the Pseudomonas Quorum Sensing, the Non Photochemical Quenching (NPQ) phenomenon [27] and the *lac* operon gene regulatory mechanism in the glycolytic pathway of *Escherichia coli* [9, 8]. Each of these models, is based on the definition of a different MP system whose evolution in time presents the same behavior of the biological dynamics under examination.

In the following we focus on MetaPlab, a software conceived and developed by Prof. Vincenzo Manca, Dr. Luca Bianco, Alberto Castellini, Dr. Giuditta Franco, Luca Marchetti and Roberto Pagliarini at the Computer science department of the University of Verona, with the support of the Computational BioMedicine Center (CBMC). In the following we will not explain the theory behind the software but we only define MP systems and employ the graphical representation of MP graphs in order to avoid complex mathematical formalisms. Detailed mathematical justifications can be found in works listed in the Bibliography section, at the end of this document.

1.1 Metabolic P Systems

MP systems are deterministic P systems where the transition to the next state is calculated according to a *mass partition strategy*, that is, the available matter of each substance is partitioned among all reactions which need to consume it. The policy of matter partition is regulated at each instant by *flux regulation maps*, or simply *flux maps*.

The notion of MP system we introduce here generalizes that one given originally in [17]. A discrete dynamical system is specified by a set of states and by a discrete dynamics on them, that is, by a function from the set $\mathbb{N}$ of natural numbers to the states of the system [26]. In this context, the natural numbers which are argument of dynamics are called *instants* or *steps*.

Definition 1 *A reaction r over substances X is represented by a pair of elements (separated by an arrow) $\alpha_r \rightarrow \beta_r$ where α_r and β_r are strings over the alphabet X . Given a symbol x , we denote by $|\alpha_r|_x$ and by $|\beta_r|_x$ the number of occurrences of the symbol x in α_r and β_r respectively. The **stoichiometric matrix** $\mathbb{A}$ of a set R of reactions over a set X of substances is $\mathbb{A} = (\mathbb{A}_{x,r} \mid x \in X, r \in R)$ where $\mathbb{A}_{x,r} = |\beta_r|_x - |\alpha_r|_x$. The set of reactions having the substance x as a reactant is $R_\alpha(x) = \{r \in R \mid |\alpha_r|_x > 0\}$ and the set of rules consuming*

or producing x is $R(x) = \{r \in R \mid \mathbb{A}_{x,r} \neq 0\}$. Two reactions r_1, r_2 **compete** for $x \in X$ if $r_1, r_2 \in R_\alpha(x)$ for some substance $x \in X$.

Definition 2 An **MP system** is a discrete dynamical system specified by a construct

$$M = (X, R, V, Q, \Phi, \nu, \mu, \tau, q_0, \delta)$$

where X, R, V are finite disjoint sets, and the following conditions hold, with $n, m, k \in \mathbb{N}$:

- $X = \{x_1, x_2, \dots, x_n\}$ is the set of **substances** (the types of molecules);
- $R = \{r_1, r_2, \dots, r_m\}$ is the set of **reactions** over X , that is, pairs (in arrow notation) of type $\alpha \rightarrow \beta$ with α, β strings over the alphabet X ;
- $V = \{v_1, v_2, \dots, v_k\}$ is the set of **parameters** (such as pressure, temperature, volume, pH, ...) equipped by a set $\{h_v : \mathbb{N} \rightarrow \mathbb{R} \mid v \in V\}$ of **parameter evolution functions**;
- Q is the set of **states**, that is, the functions $q : X \cup V \rightarrow \mathbb{R}$ from substances and parameters to real numbers. We denote by $q|_X$ the restriction of q to substances, and by $q|_V$ its restriction to parameters.
- $\Phi = \{\varphi_r \mid r \in R\}$ is a set of **flux maps**, where the function $\varphi_r : Q \rightarrow \mathbb{R}$ states the amount (moles) which is consumed/produced, in the state q , for every occurrence of a reactant/product of r . We set by $U(q) = (\varphi_r(q) \mid r \in R)$ the **flux vector** at state q ;
- ν is a natural number which specifies the number of molecules of a (conventional) mole of M , as its **population unit**;
- μ is a function which assigns, to each $x \in X$, the **mass** $\mu(x)$ of a mole of x (with respect to some measure unit);
- τ is the **temporal interval** between two consecutive observation steps;
- $q_0 \in Q$ is the **initial state**;
- $\delta : \mathbb{N} \rightarrow Q$ is the dynamics of the system, given by: $\delta(0) = q_0$ and

$$\begin{aligned} \delta(i+1)|_X &= \mathbb{A} \times U(\delta(i)) + \delta(i)|_X \\ \delta(i+1)|_V &= h_v(i+1)|_{v \in V} \end{aligned} \tag{1}$$

where $\mathbb{A}$ is the stoichiometric matrix of R over X , and $\times, +$ are the usual matrix product and vector sum.

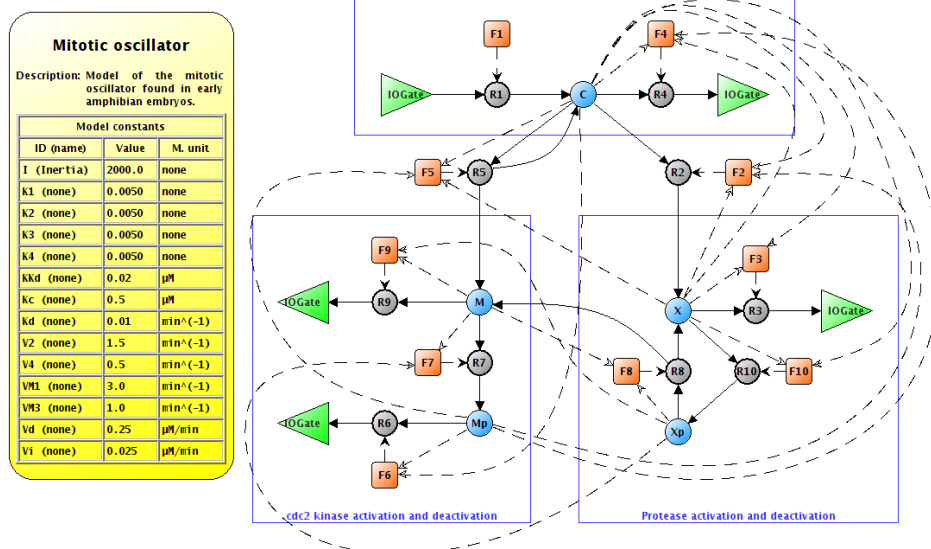


Figure 1: An MP graph representation of the mitotic oscillator model devised by Goldbeter [15, 17].

1.2 Metabolic P graphs

In order to allow a better and simpler understanding of the behavior of MP systems, the graphical formalism of MP graphs¹, has been recently introduced [23]. An MP graph is a graphical representation of MP systems by means of bipartite graphs having two levels, in which the first level describes the *stoichiometry* of reactions, while the second level expresses the *regulation*, which tunes the flux of every reaction (i.e., the quantity of chemicals transformed at each step) depending on the state of the system (see for example Figure 1). As shown in Figure 2 seven types of node and three types of edge are employed:

1. **Organism node** (rounded corner box): it stores general information of a model specifying title, description, constants and measure units of the system. Each MP graph must have one and only one Organism node.
2. **Substance nodes** (blue circles): they represent substances specifying their name, concentration values and molar weight.

¹Similar graphical formalisms were developed in the context of complex reaction networks SNA, *Stoichiometric Network Analysis*, and MCA, *Metabolic Control Analysis* (see also [28]).

3. **Reaction nodes** (black circles): they represent reactions.
4. **Input gates** (triangles with an edge at a vertex): they are connected to reaction nodes which specify rules that do not respect the *Lavoisier Principle* [17] because they introduce new matter into the model.
5. **Output gates** (triangles with an edge at a basis): they are connected to reaction nodes which specify rules that consume matter.
6. **Flux nodes** (rounded corner boxes): they represent fluxes of matter transformed by reaction nodes. Each reaction node is connected to one and only one flux node containing the evolution function which regulates the reaction flux at each step.
7. **Parameter nodes** (rectangle boxes): they represent parameters. Each parameter evolution can be defined by means of an evolution formula or a time-series.
8. **Reaction edges** (arrows): they connect Substance nodes to Reaction nodes specifying *reactants* and *products* of each reaction.
9. **Regulation edges** (dashed arrows): they connect Flux nodes to Reaction nodes in order to represent the regulation of each rule.
10. **Dependence edges** (dashed arrows): they connect Substance nodes or Parameter nodes to Parameter nodes or to Flux nodes whose evolution formula refers to them.

In Figure 1 is depicted the MP graph obtained by modelling, by MP systems, the Golbeter's differential model of mitotic oscillations [15, 17].

2 MetaPlab: a brief introduction

MetaPlab [6] is a Java software which intends to assist biologists to understand the internal mechanisms of biological systems and to reproduce and analyze, in silico, biological phenomena: responses to external stimuli, environmental condition alterations, and structural changes. It extends an MP systems simulator, called *Psim* [1, 5]. The latest version of MetaPlab is available at the MetaPlab website:

<http://mplab.sci.univr.it>.

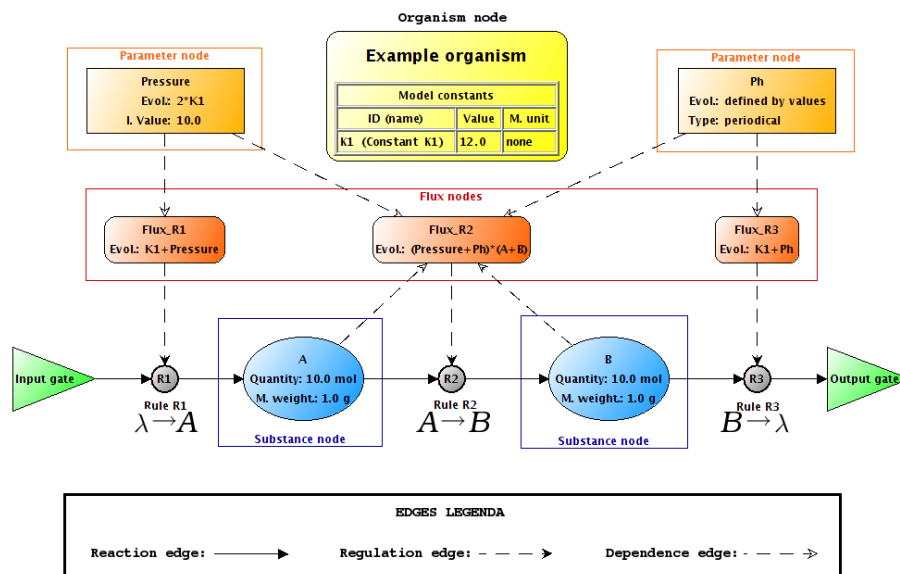


Figure 2: A toy MP graph with a legenda of different nodes and edges.

and it requires Java 1.6 (or later versions) installed on the user’s computer in order to be run. The Java implementation of MetaPlab ensures the cross-platform portability of the software, which is released under the GPL open-source license.

2.1 Plugin framework

The new computational framework we propose is based on an extensible set of *plugins*, namely Java tools for solving specific tasks relevant in the framework of MP systems. Among these tasks, regulation function discovery, simulation, visualization, graphical and statistical curve analysis, importation of biological networks from on-line databases, and possibly other aspects result to be significant for further investigations. Figure 3 depicts this framework, which involves four main layers: the first deals with the model definition and visualization by *MP graphs*, the second is dedicated to the representation and storing of MP systems by a suitable data structure called *MP store*, the third concerns with the processing of these data by means of computational units called *MP plugins*, and finally, the fourth arranges a set of *vistas* which support the MP systems analysis. The new extensible data processing layer makes MetaPlab a proper “virtual laboratory” wherein MP plugins act as virtual tools for processing MP systems.

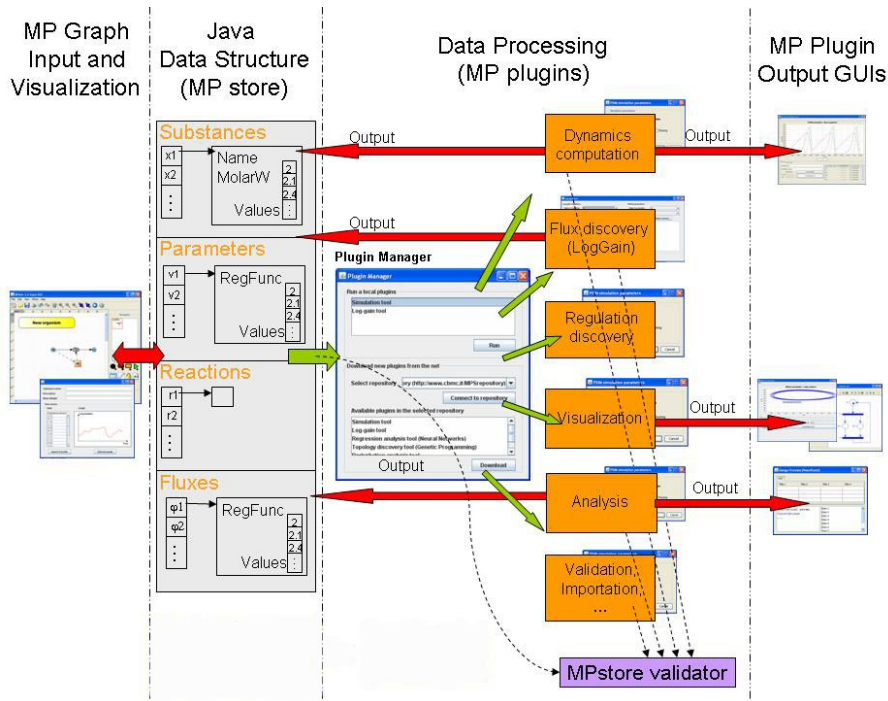


Figure 3: The MetaPlab framework.

Some plugins are distributed along with the MetaPlab main package while further plugins (or update versions) can be downloaded from the plugins page of the MetaPlab website

<http://mplab.sci.univr.it/plugins/Plugins.php>.

The downloading procedure consists of selecting the *Download* link for a specific plugin and then choosing a suitable directory where storing the archive. The uncompressed plugin file has finally to be copied into the *pluginsExt* directory of the MetaPlab package in order to be automatically loaded at the start up.

2.2 MetaPlab input GUI

Figure 4 shows the input GUI of MetaPlab with some magnifications of the most important sections. The input GUI is displayed when MetaPlab is launched by clicking on the MetaPlab.jar file or on one of the batch files (called MetaPlab-windows-xxxM or MetaPlab-linux-macOS-xxxM) that enable to allocate specific amounts of memory to the software. The input GUI

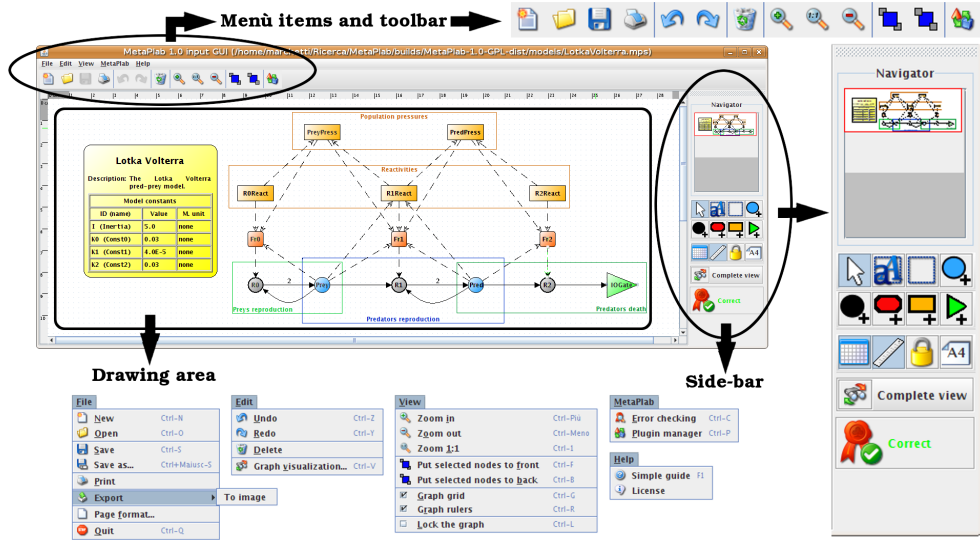


Figure 4: MetaPlab input GUI.

is divided into three different areas which includes menus, a top-toolbar, a side-bar and a central drawing area where MP graph can be depicted. At the bottom of Figure 4 all the menu items of this window are reported. Many of these commands are replicated in the top-toolbar or in the side-bar in order to speed up the modelling procedure. Particular care must be dedicated to controls collected in the side-bar because they have to be used in order to generate new graphs.

3 Designing an MP model

In this section we create an MP graph by means of the MetaPlab input GUI. The model that we consider for this example is an oscillator called Sirius (creativus) (a variant of Sirius model proposed in [19]) involving three substances:

A : initial concentration : 100 moles	molar weight : 1.0 gram
B : initial concentration : 100 moles	molar weight : 1.0 gram
C : initial concentration : 0.02 moles	molar weight : 1.0 gram

and five reactions whose fluxes are tuned by the following regulation func-

tions, where I is a constant and its value is 100:

$$\begin{array}{ll}
R0 : A \longrightarrow A^2 & F0 : \frac{10A}{I + 10 + 0.02C + 0.02B} \\
R1 : A \longrightarrow B & F1 : \frac{0.02CA}{I + 10 + 0.02C + 0.02B} \\
R2 : A \longrightarrow C & F2 : \frac{0.02BA}{I + 10 + 0.02C + 0.02B} \\
R3 : B \longrightarrow \lambda & F3 : \frac{4B}{I + 4} \\
R4 : C \longrightarrow \lambda & F4 : \frac{4C}{I + 4}
\end{array}$$

3.1 Adding nodes

After having launched MetaPlab, we start to generate our model. The drawing area of the input GUI is never completely empty because an Organism node is always displayed. This node collects the general information about the current model, such as, its name, its constants and its measure units. We begin to draw our model by editing the default information stored into the organism node. To do this we either double-click on the node or select the node by a single click and then start the editing procedure by pressing the F2 button of the keyboard¹.

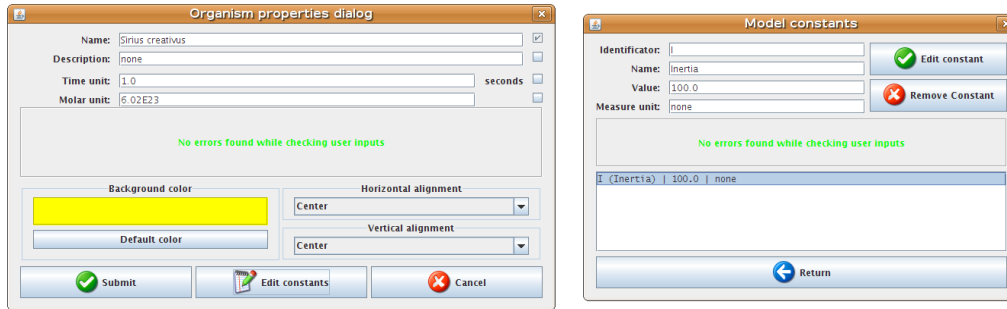


Figure 5: The windows employed to edit the Organism node content (on the left) and the list of the model constants (on the right).

When we begin to edit the node, a window is visualized as in the left part of Figure 5 and we can change the default values by typing new values, as

¹This procedure is general and can be used, in the following, to edit the data of each type of node.

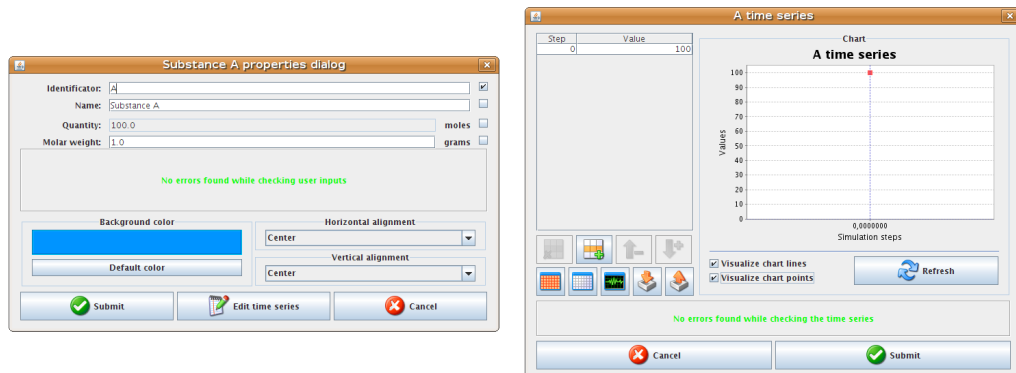
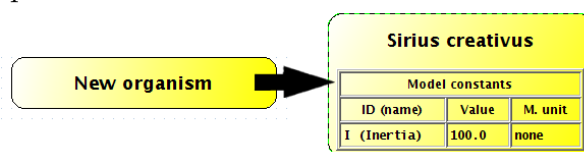


Figure 6: The windows to edit the Substance node content (on the left) and the time-series of concentration values (on the right).

shown in the left side of Figure 6. In the window each text-box is associated to a small check-box visualized on its right side. This check-box can be selected in order to force the visualization of the data into the MP graph. In the center part of the window there is a framed text-box which prompts an error message if we insert wrong values (for example if we insert a textual value where the application needs a number). Using this window we can also start to edit the model constants list in order to add the constant I defined before. To do this we click on the button **Edit constants** and then we use the window subsequently displayed, as shown in the right side of Figure 5.

After having changed the Organism node values, we submit them by clicking on the **Submit** button. The Organism node updates its content as in the following picture:



Now we add to the drawing area three Substance nodes needed to define the stoichiometry of the system. To do this we click on the button identified by a blue circle icon in the side-bar of Figure 4. When we do this, the mouse pointer changes its shape because the software has entered into the *insertion mode*. At this point we can add Substance nodes by moving the pointer to the position where we want to create the new node and then clicking by the left button of the mouse. After the three Substance nodes have been created, we exit from the insertion mode by clicking on the side-bar button having an arrow as icon. Now we can change the default values of the nodes

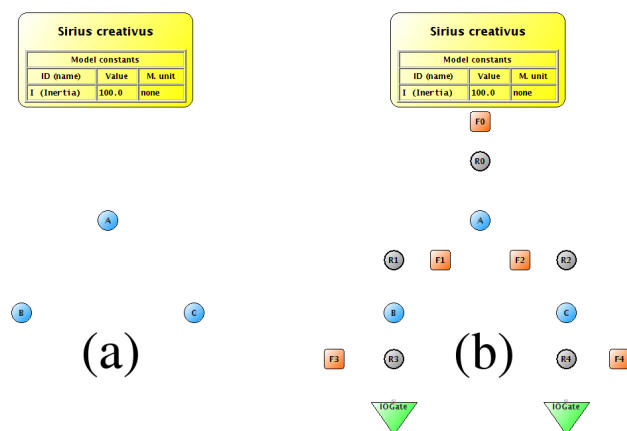


Figure 7: Sirius at different modelling steps: after the insertion of Substance nodes (a) and after the insertion of Reaction, Flux and Gate nodes (b).

just created by the window depicted in the left part of Figure 6¹. In order to edit the initial concentration of each Substance node, we use the window depicted in the right part of figure 6 which is visualized by clicking the **Edit time-series** button. This second window has some commands which enable to edit and plot time-series of concentration values. In our case we only need to update the initial concentrations by the values given at the beginning of this section. The position of each Substance node can be changed dragging and dropping the related graphical elements obtaining the graph of Figure 7a.

In order to complete the stoichiometry of the system, now we need to add five Reaction nodes which represent the five rewriting rules defined at the beginning of this section. To do this we need to click on the button of the side-bar having a black circle as icon, and to pass again to the insertion mode. In the same way we add five flux nodes and two gate nodes by clicking, respectively on the buttons of the side-bar having a red rectangle or a green triangle as icon. We finally exit from the insertion mode, as usual, and we drag nodes to positions showed in Figure 7b.

Now let we edit the data collected in each new node by the two windows displayed in Figure 8. They can be opened, as usual, by double-clicking on a node or selecting a node by a single click and then starting the editing procedure by pressing the F2 button of the keyboard. The first interface enables to modify properties and appearance of a gate node² while the second

¹In order to visualize the window, please use the same procedure described to edit the Organism node.

²An important feature of this window is the possibility to change the gate orientation.

one permits to update the data of a Flux node.

3.2 Drawing edges

After having updated the properties of each node by the Sirius data, presented at the beginning of this section, we can start to create the edges required to complete the MP graph definition. The creation of dependence edges is automatic when we enter an evolution formula for a flux node. Each new edge is initially displayed by a straight line, but its appearance can be changed by adding some *control points* and by dragging them to different positions of the drawing area (as displayed in Figure 10). We can add or remove a control point to an edge by clicking with the right button on the edge that we want to modify. Bent edges if Figure 9a have been created by using control points.

Now it is time to complete our model by adding reaction edges and regulation edges. The creation of an edge is easy to be perform in MetaPlab. In Figure 11 it is depicted a magnification¹ of some MP nodes where we can distinguish some small rectangular areas². These areas are called *ports*. If we want to create a new edge we have to place the mouse pointer over the port of its source node, click on it when the mouse pointer changes its shape, and drag the mouse pointer to the destination node. The creation of the edge is automatic, thus the user does not need to change the appearance of an edge by hand because the software is able to understand which type of edge has to be generated and it changes accordingly its appearance. When we create

¹Please remember that we can always zoom in or zoom out the graph using the commands of the toolbar or of the **View** menu.

²If you are not able to see the port of a node, please try to zoom in the graph and check that the graph is unlocked. You can lock or unlock the graph by using the button of the side-bar with a lock as icon or the last command of the **View** menu.

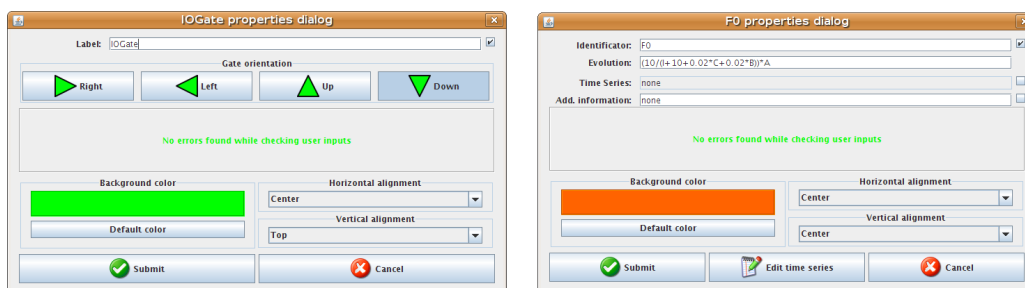


Figure 8: The windows to edit the properties of an Input/Output gate (on the left) and of a Flux node (on the right).

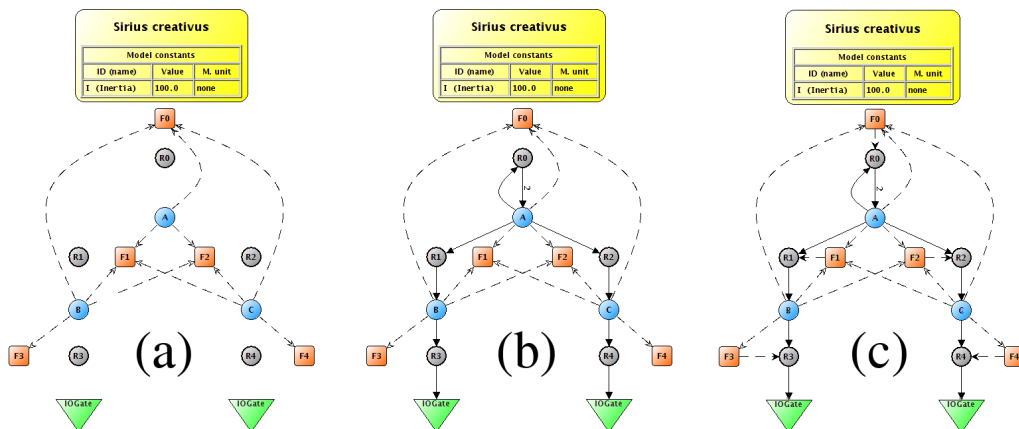


Figure 9: The Sirius model at different modelling steps: after the editing of Flux nodes (a), after the insertion of Reaction edges (b) and, finally, the complete model (c).

the reaction edges for rule $R1$, we need to change the multiplicity of the edge which connects node $R1$ to node A ¹. To do this we double-click on the edge and then we change the value using the window depicted in Figure 12. The model is now completed and its graphical aspect is shown in Figure 9c.

3.3 Checking model correctness

During the modelling procedure the application visualizes a small framed label at the bottom of the side-bar which checks the correctness of the model. If any mistake occurs, it is possible to visualize a window which prompts a list of errors that must be eliminated in order to use the model (Figure 13). To visualize the window, we click on the framed label of the side-bar. If we correctly followed all the modeling steps the MetaPlab checker confirms the model correctness, otherwise a list of errors is prompted and the user

¹Please remember that each reaction edge which does not display the number of its multiplicity has this value set to 1 (the default value).

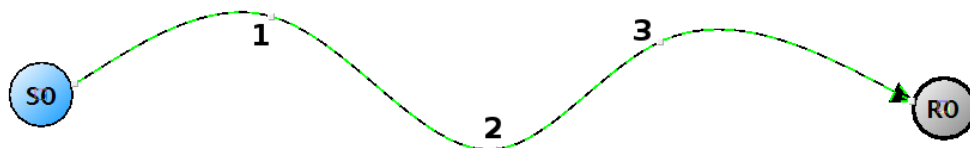


Figure 10: A magnification of a reaction edge with three control points.



Figure 11: A magnification of MP nodes where it is possible to see the port of each node. Each Gate node (the last on the right in the picture) has two different ports: the first one (on the left) must be used to define output rules while the other one is for input rules.

has to correct them before launching any plugin¹. When the model is finally completed and correct, we can save it.

4 Simulating and plotting an MP system behavior

After generating a new MP graph and obtaining the correctness confirmation from the MetaPlab checker, many processing tools can be run. We start explaining a plugin for computing the MP model dynamics and, then, a second tool which enables the user to plot and analyze dynamics charts.

¹Please remember that when we edit a model, we can always come back to its previous versions by using the undo/redo support given by MetaPlab.

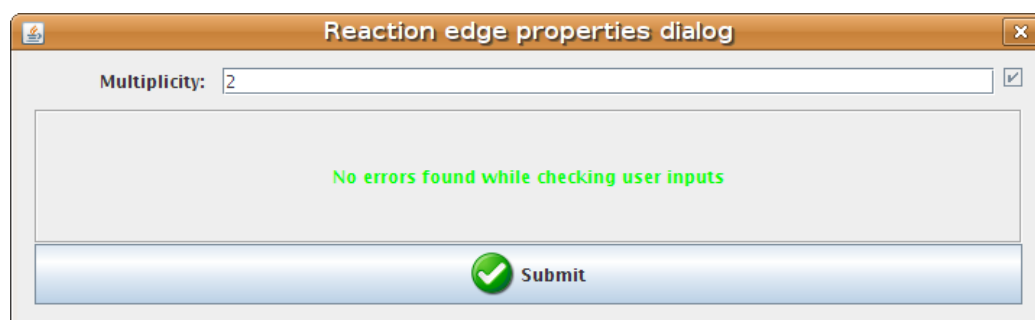


Figure 12: The window which permits to change the multiplicity value of a reaction edge.

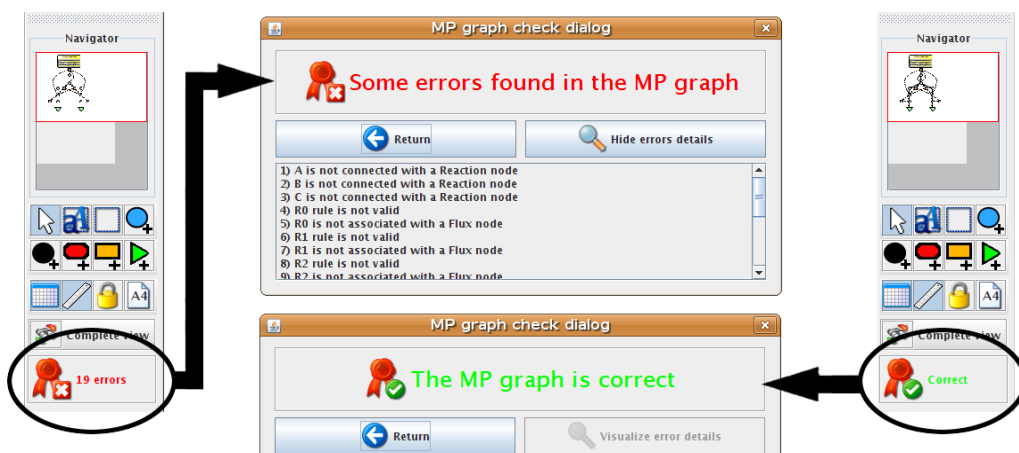


Figure 13: The checking of the model's correctness by MetaPlab.

4.1 Dynamics computation plugin

In order to process MP models we employ plugin tools performing various tasks. We launch the plugin manager by clicking on the rightmost button of the top-toolbar, highlighted in Figure 14, or by using the related command of the **MetaPlab** menu. The plugin manager of MetaPlab is the tool which permits to get advantage of the plugin architecture of the software introduced in [6]. Whoever can write a plugin in order to perform a wide variety of operations on MP model data. In this section we employ two plugins which are distributed within the software. The first one is called *dynamics computation* and it is used to compute the dynamics of the model currently loaded. To launch the plugin we select its entry in the plugin list prompted by the plugin manager and then we launch it by pressing the **Run** button. Before computing the dynamics, the plugin visualizes a window where the user can insert some simulation parameters, such as the number of simulation steps which we set to 1000. Then we start the simulation by pressing the **Start** button. After few seconds (execution time depends on the number of steps performed), the simulation procedure ends and we can go back to the plugin manager by clicking on the **Return** button. All the steps described before are depicted in Figure 14. The simulation plugin of MetaPlab offers to the user many functionalities which can be used to manage the simulation process. Among them we point out the possibilities of:

- stopping or resuming the dynamics computation by using the **Start** and **Stop** buttons;
- changing the initial step from which to start the computation of the

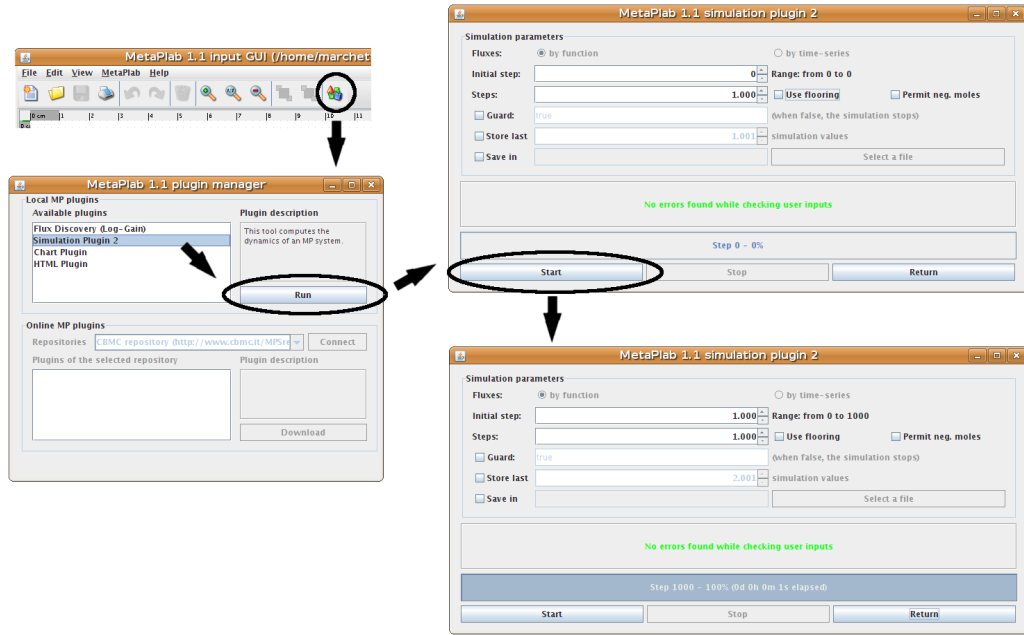


Figure 14: The procedure to simulate the dynamics of an MP model.

dynamics, if possible;

- setting the number of steps to compute;
- defining a boolean expression which must be true during the simulation procedure (when it becomes false, a window is prompted to the user and the simulation is paused);
- changing the number of steps which the user wants to store in memory (if this number is less then the number of simulation steps, then only the last values will be maintained in memory)¹;
- storing to file the time-series computed by the software.

4.2 Chart plotting plugin

When the dynamics of our Sirius model have been computed, we plot it by means of another plugin called **ChartPlugin** which can be launched at the

¹This functionality can be useful when we want to investigate the behavior of an oscillatory phenomenon because it permits to run long simulations without needing a big quantity of memory.

same way of the previous one (Figure 15). This plugin visualizes a window which enables to create different types of chart. The plugin window is divided in two parts: in the upper side it is depicted the chart that the user wants to study, in the bottom side there are three panels which enables to create line charts, phase charts and to set some general options.

First of all, we create a line chart which displays the concentration of the three substances of the model. The result of this operation is depicted in the right side of Figure 15. Moreover we can create phase charts using the second panel of the plugin window as depicted in Figure 16. Charts created by the plugin can be exported to file, printed and zoomed. In order to reduce memory load, the creation of each chart is performed by means of an algorithm which uses data sampling. In the third panel it is possible to set a parameter which modifies the accuracy of the chart plotting. Its default value is set to 1, but it can be changed to any rational number¹. Figure 17 displays the same line chart plotted using different parameter values for data sampling.

4.3 Result analysis

When we close the plugin and the plugin manager a window is prompted asking to update the current model by new values computed during the plugins execution. Confirming the operation, the update is performed in a few seconds. If we click on substance node *A*, in order to check its properties, we find the new time-series computed during the simulation, as displayed in Figure 18.

¹Values greater than 1 reduce the accuracy of the plot, while smaller values increase it.

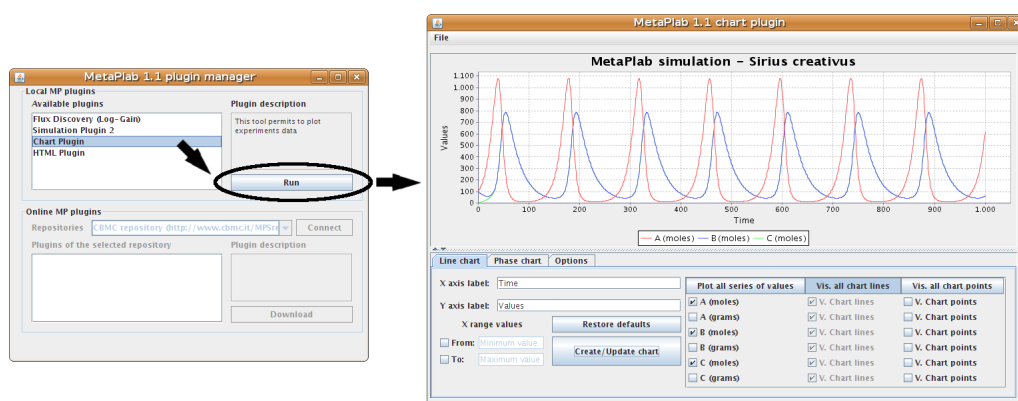


Figure 15: The procedure to display the simulation values of an MP model.

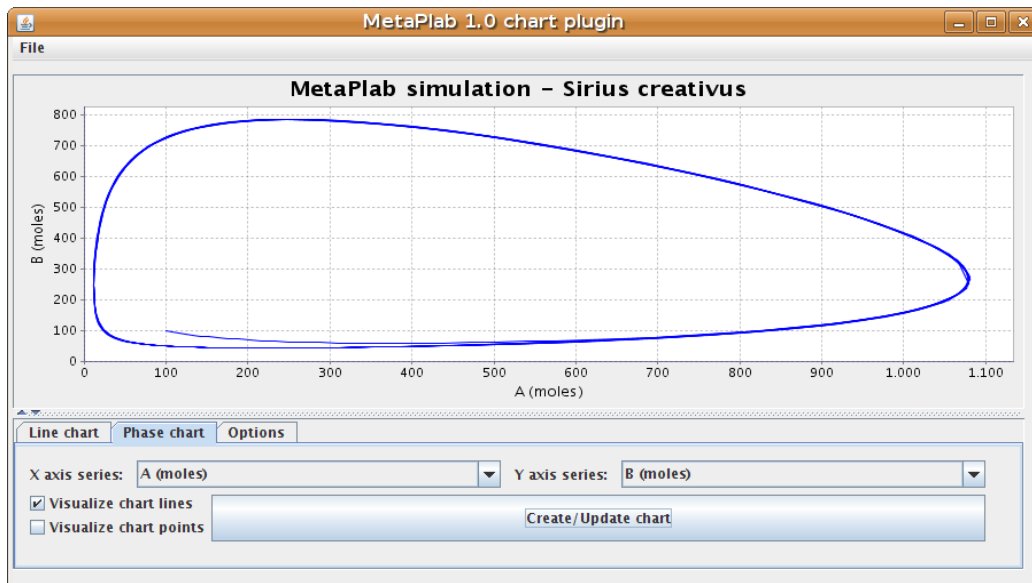


Figure 16: A phase chart of the substance A and B of the Sirius model.

Now we try to improve the readability of the MP graph by adding some frames and labels. The MetaPlab input GUI offers also the possibility to hide a part of the MP graph without deleting its nodes, as depicted in Figure 19, a useful feature for managing complex graphs. This operation can be performed by a window which can be launched by clicking on the button of the side-bar having a pair of binoculars as icon, or by selecting the right command of the **Edit** menu. The area of the MP graph to be visualized or hidden have to be selected before launching the visualization window or it can be defined specifying a particular type of node or edge.

4.4 Adding parameters to MP models

In this subsection we propose a little modification of the Sirius model where we add some random noise to flux functions. To do this we add to the model a *parameter* node N which represents the noise and then we change each flux formula to the following ones:

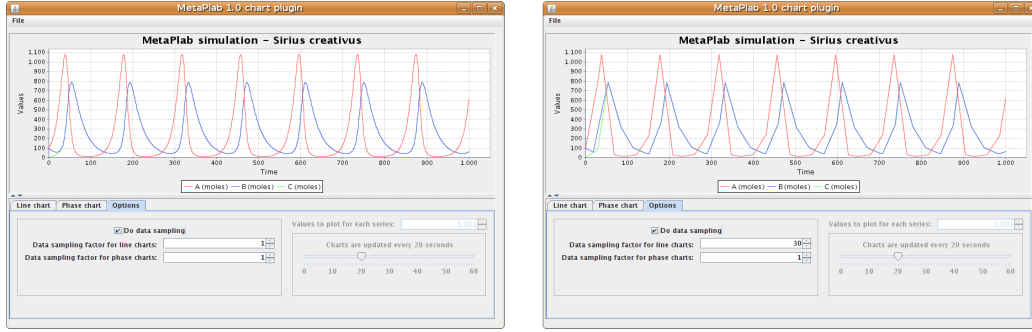
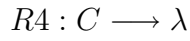
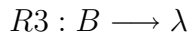
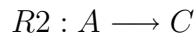
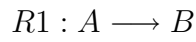
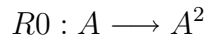


Figure 17: The same line chart depicted with different data sampling values (1 on the left and 30 on the right).



$$F0 : \frac{10AN}{I + 10 + 0.02C + 0.02B}$$

$$F1 : \frac{0.02CAN}{I + 10 + 0.02C + 0.02B}$$

$$F2 : \frac{0.02BAN}{I + 10 + 0.02C + 0.02B}$$

$$F3 : \frac{4BN}{I + 4}$$

$$F4 : \frac{4CN}{I + 4}$$

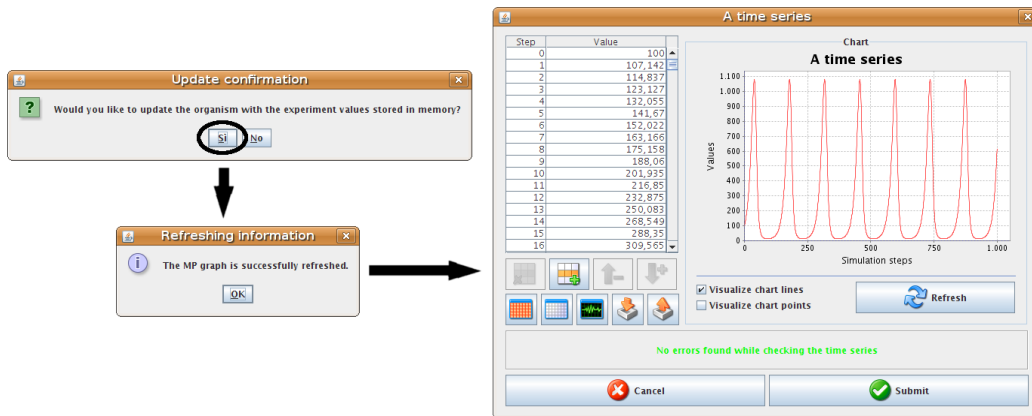


Figure 18: The update of the MP graph after the closure of the plugin manager: windows (on the left) and the new time-series for A (on the right).

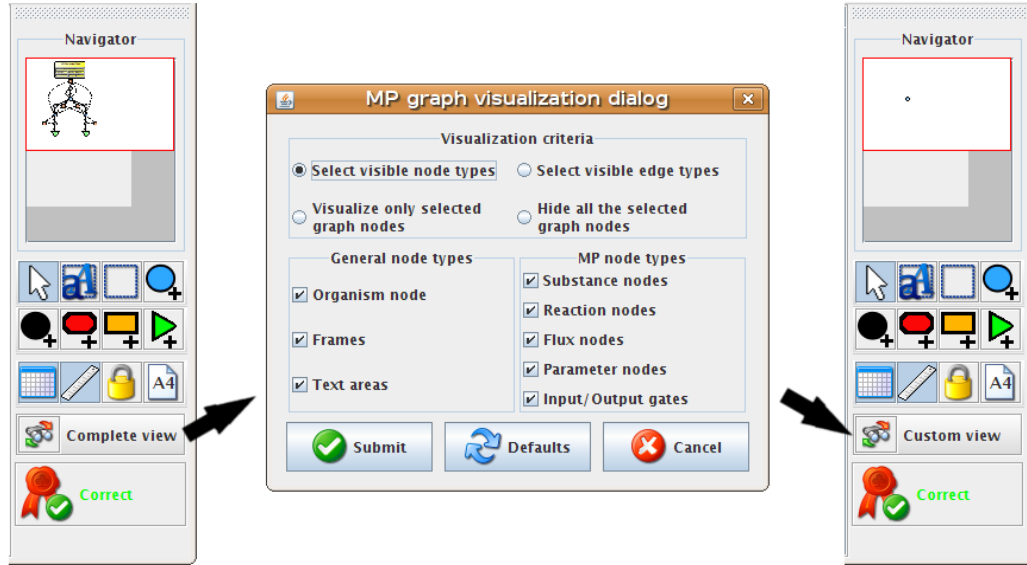


Figure 19: The window employed to change the visualization of the current MP graph.

where $I = 100$.

The first thing to do is to modify our MP graph by adding a parameter node. The button of the side-bar which enables to add parameter nodes is that having an orange rectangle as icon. The windows which can be used to edit a parameter node are depicted in Figure 20 and 21. A parameter node can be defined in two different ways: by specifying an evolution formula and an initial value, or by defining a time-series of values to be used during the dynamics computation. When in a model one or more parameter nodes are defined by values, the number of steps computed by the dynamics computation plugin is limited to the minimum length of the time-series specified for each parameter. There could be some cases, nevertheless, in which we want to specify by values a parameter that models a periodical function¹. In these cases we add the time-series of one oscillation and then we select the **Periodical parameter** checkbox in order to tell to MetaPlab of to consider the time-series of the parameter as a circular vector of datas. In these cases the number of steps to be computed by the dynamics computation plugin will be not limited and the dynamics will display a perfect oscillatory behavior.

The parameter node which we add to the MP graph in order to add a noisy signal is defined by values and set to be periodical. First of all we modify the

¹For example it could be considered as periodical a parameter which models the intensity of the light during a day.

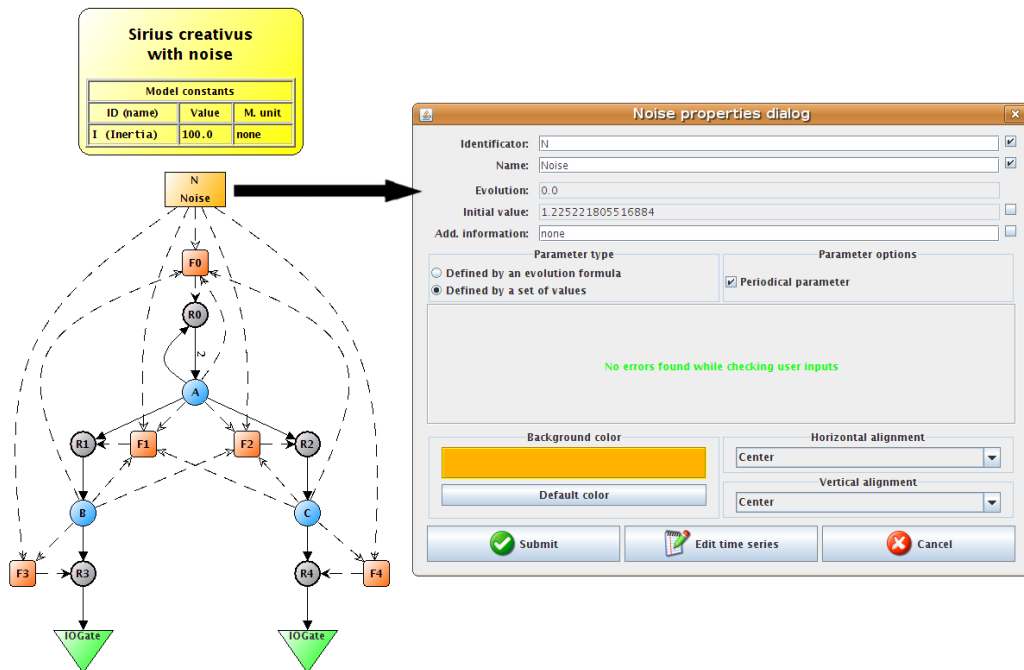


Figure 20: The MP graph of the Sirius model with noise (on the left) and the window to set the noise Parameter node (on the right).

values of the parameter as depicted in the right side of Figure 20 and then we edit the parameter time-series. To do this we have to click on the **Edit time-series** button which enables the visualization of the first window of Figure 21. It is not the first time that we use this kind of window, indeed, it is the same that we have used before to specify the initial concentration of substances, but this time we insert more values. The window is divided in two parts, in the left side there is a table where we can directly add, edit or delete each value of the parameter time-series, while in the right side it is depicted a chart of time-series. Under the table there are some buttons that enable to perform some automatic operations that can help the user when he/she has to manage a time-series having many values. For example, MetaPlab gives the possibility to import data from files or to export loaded values to file¹.

In order to model a noisy function, we need to generate random data. MetaPlab provides a generator of random values which can be started by clicking on the button having a small chart as icon. The generation window is depicted in the right side of Figure 21. It enables to generate random values

¹By now MetaPlab considers only text files which have each value in a single row without the addition of other characters.

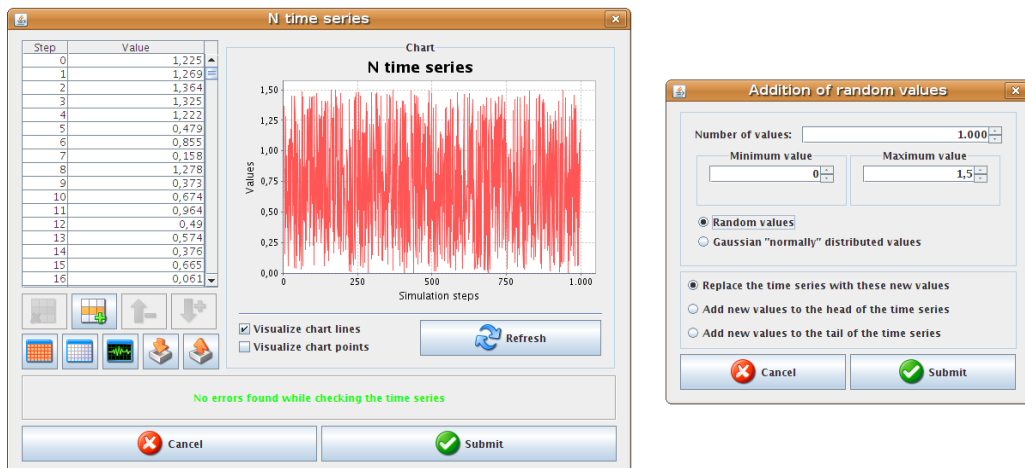


Figure 21: The windows to set the time-series of the noise Parameter node.

or Gaussian distributed values. By now we set the window as displayed in the picture and then we click on the **Submit** button. After having started the generator, the time-series updates its content and the parameter node is ready to be used. We edit the flux functions as specified at the beginning of this subsection. The final model is depicted in Figure 20.

We finally compute the dynamics of the model (as explained in the previous subsection) and we compare it to the one previously obtained without the noisy signal. Figure 22 displays the charts of the two test cases obtained by the simulation of 500 steps of the two models.

5 Log-gain plugin

By using mathematical models we can examine systems that we are unable to observe and understand directly. In the last years different approaches for constructing models of biological phenomena starting from experimental data have emerged. Their goal is to give a view of the phenomena under investigation at different organization levels and to test their responses to different inputs. The core of each procedure that generates these models is the identification of some parameters that make models consistent with observed data. MP systems provide a new theory, based on the Log-Gain Principle [20], for inferring models from experimental data. They allow to determine metabolic fluxes associated with each reaction by using algebraic manipulations of observed data.

Since flux discovery is the first step towards the generation of MP models from experimental data, MetaPlab offers a plugin based on Log-gain theory

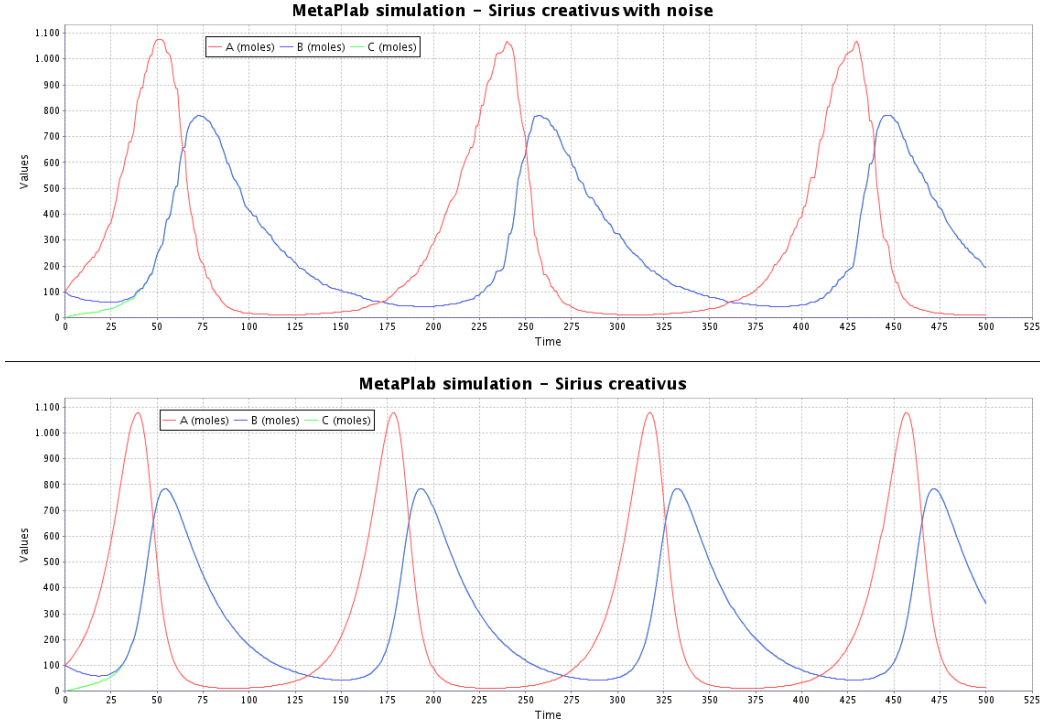


Figure 22: The charts which plot 500 simulation steps of the Sirius model with noise (on the top) and of the classical one (on the bottom).

which automates this process. In this section we describe how to use the *Log-Gain plugin* to infer fluxes related to observed dynamics. We will refer to Sirius, the synthetic oscillator already introduced in Section 3.

We suppose to know 1000 steps of substance and parameter time-series, and the initial value of each flux. We remark that the i -th value of substance x_j , parameter v_j and flux φ_{r_j} will be denoted, respectively, by $x_j[i]$, $v_j[i]$ and $u_{r_j}[i]$. Moreover, we define “tuners” or “regulators” of a reaction r as substances and parameters which influence, by their variations, the flux variations of r . The following sets of tuners have been assigned to Sirius by the analysis of its structure. Here we don’t enter into the details of tuners discovery:

$$\begin{aligned} T_{R0} = T_{R1} = T_{R2} &= \{A, B, C\} \\ T_{R3} &= \{B\} \\ T_{R4} &= \{C\} \end{aligned}$$

The Covering Offset Log Gain Property (the reader can find more details

about this property in [20]) establishes that a rule of $R(X)$ - rules where X occurs - has to be assigned to ant substance X . In our case, let us provide the following covering: $\mathbb{R}_0 = \{R0, R1, R2\}$, with $R0 \in R(A), R1 \in R(B), R2 \in R(C)$. Tuners and covering are the inputs required by the Log-gain plugin, along with substance and parameter time-series, for inferring flux time series $u_{T_j}[i], j = 1, 2, \dots, m$.

Loading initial fluxes and time-series. Before starting the plugin we load observed substance and parameter time-series, as explained in Section 3, and we also insert the initial value of each flux, which is fundamental to generate flux time-series by the Log-gain theory. We then launch the plugin manager by clicking on the right button of the MetaPlab top-toolbar or by using the corresponding command of the MetaPlab menu (this procedure has been explained in Section 4). To launch the plugin we select *Flux discovery (Log-Gain)*, as showed in Figure 23, in the list prompted by the plugin manager and then we click on the *Run* button. After these steps, the plugin GUI is visualized, as shown in Figure 24. It is essentially divided in two parts:

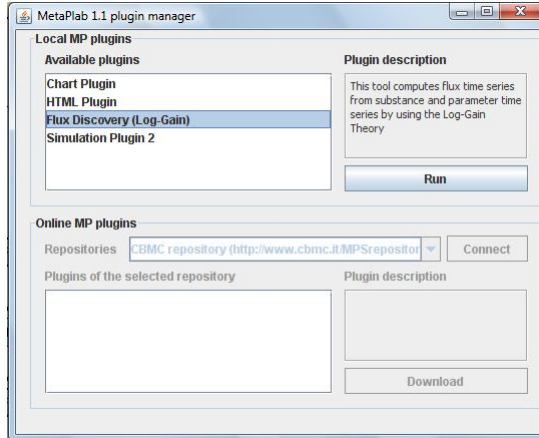


Figure 23: Launching the Log-gain plugin.

in the left side we assign a set of tuners to each reaction, while in the right side we set up the set of reactions which satisfies the covering property of the phenomenon under investigations.

Assigning tuners to reactions. Let us suppose to want to assign the substance A to the reaction $R0$ in the left part of the plugin interface. We select entry $R0$ from the upper drop-down list and the entry *Substance A* from the lower list and then we click on the *Add regulator* button. After these steps, the entry $R0 - A$ is added to the list panel below. Figure 25

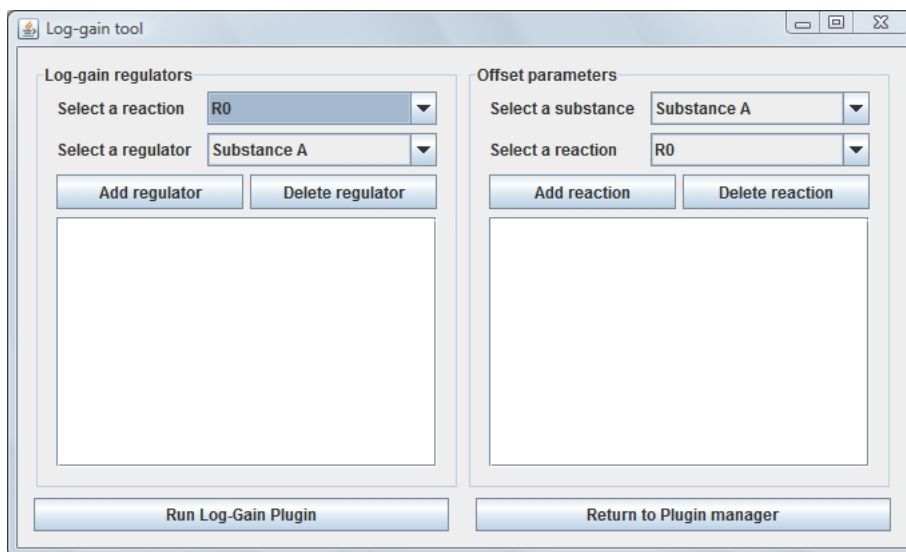


Figure 24: The main graphical interface of the Log-gain plugin.

graphically shows how the plugin GUI is updated after each step. By following the same steps we add tuners to every reaction. Figure 26 shows, in the left side, the list of all the tuners of each reaction.

Assigning covering reactions. After the tuner definition, we move to the right side of the interface in order to insert the set of reactions which has the Covering Offset Log Gain Property, that is, the reactions that belong to the set $\mathbb{R}_0$. The procedure is very similar to that performed to insert tuners. In this case, we cover every substance by a reaction of $\mathbb{R}_0$. Let us suppose we want to insert the reaction R_0 , which covers the substance A . We select entry R_0 from the upper drop-down list and entry *Substance A* from the lower drop-down list and then we click on the *Add reaction* button. After these steps, graphically represented in Figure 27, the entry $R_0 - A$ is added to the list panel below. In the same way we add reactions R_1 and R_2 which cover, respectively substances B and C . Figure 28 shows the list containing the complete set $\mathbb{R}_0$.

Deleting wrong tuners and/or wrong covering reactions. Let us suppose to have inserted a wrong tuner, such as, substance A as tuner for reaction R_3 . In this case, we can delete entry $R_3 - A$ from the corresponding list in a very simple way. It is sufficient to select this couple from the list and click on the *Delete regulator* button. This step is depicted in Figure 29. In the same manner it is possible to delete a covering reaction from the list

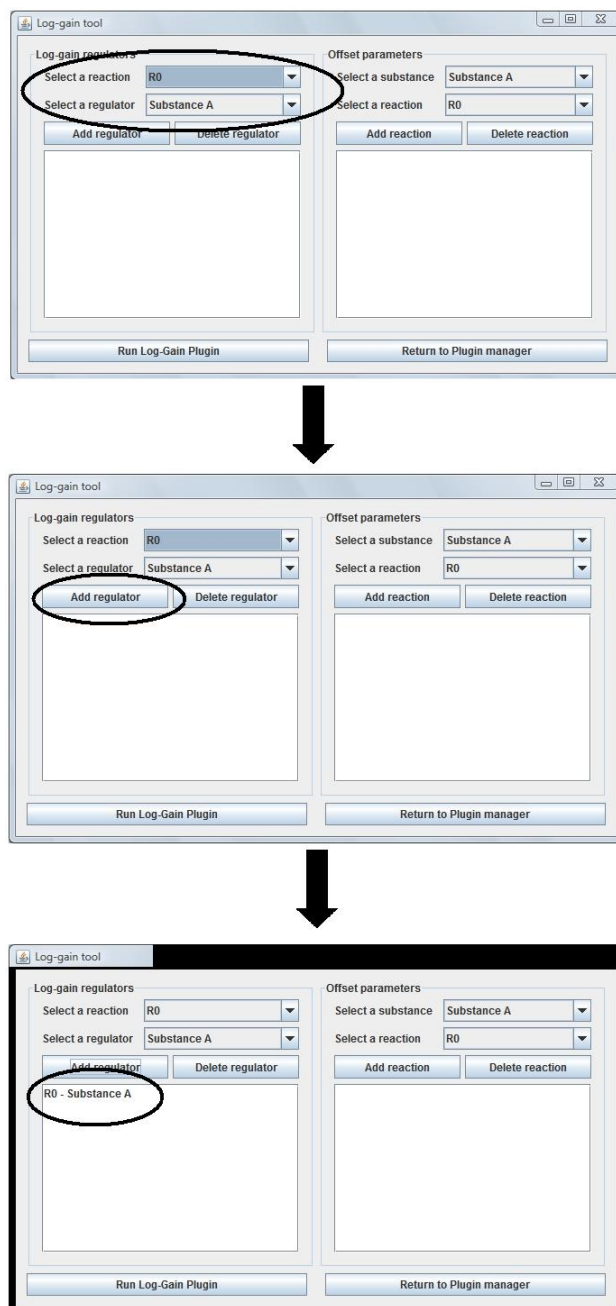


Figure 25: Adding tuners to reactions: i) a reaction-tuner entry is selected; ii) the *Add regulator* button is clicked; iii) the couple is inserted into the list.

on the right side.

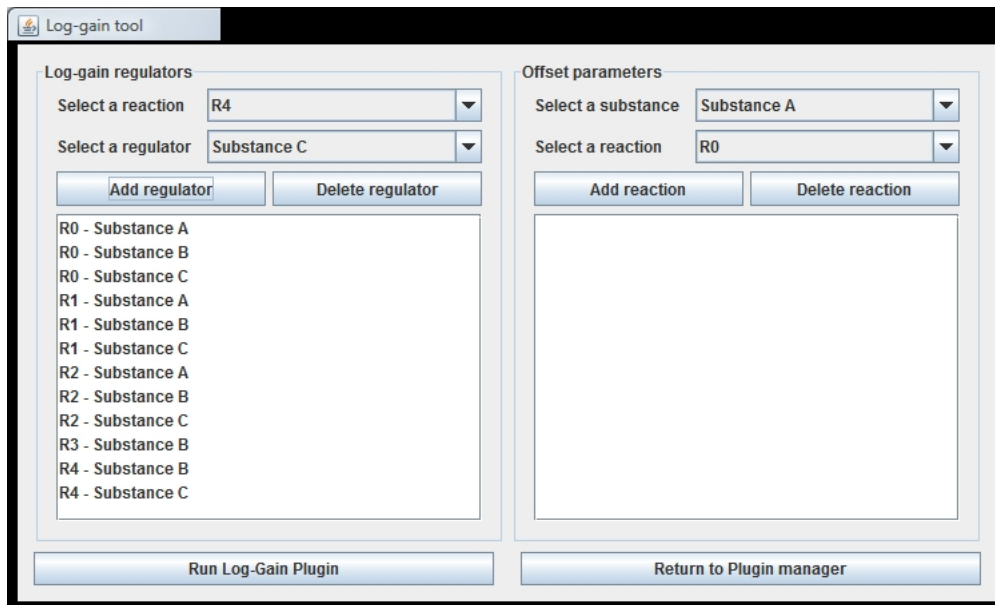


Figure 26: Sirius tuners.

Computing the flux time-series. If all the tuners and the covering reactions are correctly inserted, then we can compute the flux time-series of every reaction by clicking on the *Run-Log Gain Plugin* button (see Figure 30). The plugin starts its computation and, after few seconds (this time depend on the dimension of the problem and on the quantity of data used), it returns an information message which announces the end of the computation. At this time, flux time-series are automatically returned to the plugin manager.

In order to complete our example, it is possible to see the results obtained by the Log-Gain plugin in Figure 31. Fluxes inferred from substances time-series by the Log-Gain plugin are almost identical to flux time-series generated in Section 3 during the dynamics computation. It is always possible to return to the plugin manager by clicking on the button *Return to Plugin manager* (see Figure 32).

Log-Gain plugin requires that a few conditions related to the Log-Gain Principles be satisfied, otherwise, the software returns warning and error messages. The main warning messages are displayed when:

- the initial value of some fluxes has not been defined into the MP graph. In this case, in order to infer the time series of fluxes, the user must return to the MP graph and inserts these fluxes values;
- the covering of substances by reactions is not complete and the user

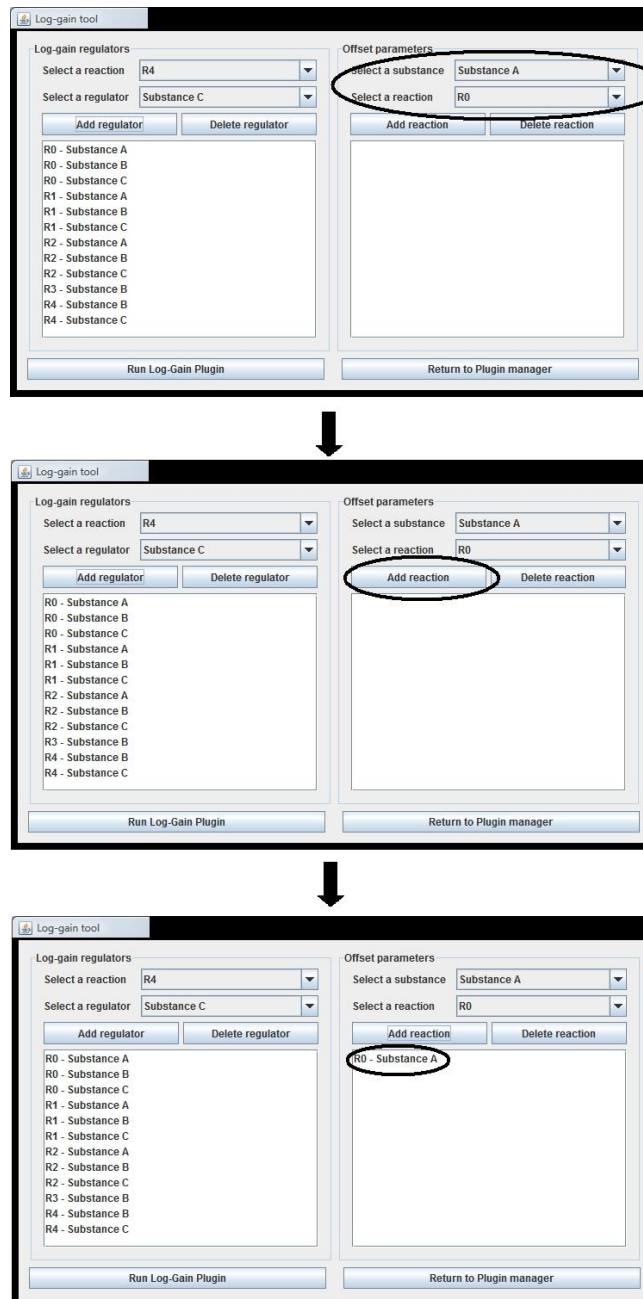


Figure 27: Adding reactions which fulfill the Covering Offset Log-Gain Property: i) a reaction-tuner entry is selected; ii) the *Add reaction* button is clicked; iii) the entry is inserted into the list.

click on the *Run Log-Gain Plugin* button.

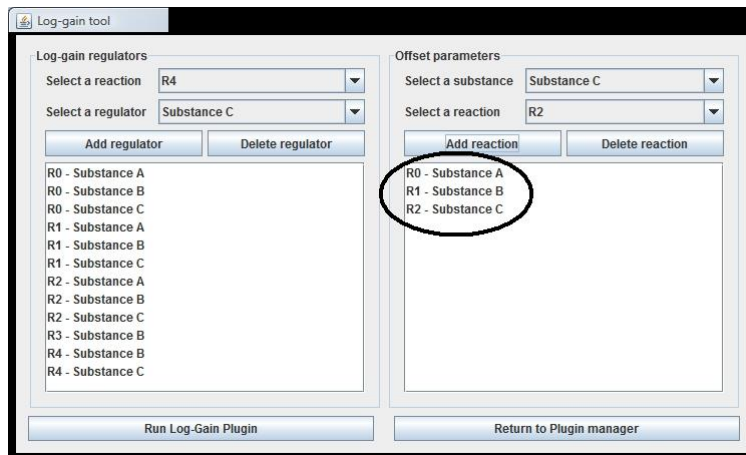


Figure 28: Covering set $\mathbb{R}_0$ for Sirius.

6 Linear regression plugin

In this section some basic concepts are reported which are necessary to understand how the *Linear Regression tool* works and the theory behind it: we firstly introduce the problem of discovering regulation functions in MP systems, then *linear regression* is presented, and finally we discuss the main features of the linear regression tool which can be employed in the context of *MetaPlab* software, to automatically generate regulation functions from data.

A main task of MP modeling process concerns the synthesis of MP regulation functions from experimental data. Nowadays biologists have several high and low-throughput experimental techniques able to provide time-series of substance quantities and chemo-physical parameters. These data are employed into MP models for regulation functions. Given some substance and parameter time-series, collected by experimental observations, two main steps have to be performed in order to infer flux regulation functions: *i)* to compute flux time-series from substance and parameter time-series, *ii)* to synthesise regulation functions from substance, parameter and flux time-series. The *Log-gain theory* [19] supports the first step while the second one requires a regression technique to compute sound regulation functions from observed data.

Linear regression plug-in is a tool devised to automatically synthesize MP regulation functions from time-series of substances, parameters and fluxes by means of linear regression. It takes as input both observed data and a parametric form of every regulation function and it returns as output a set of regulation functions that fit observed data. Polynomial functions have to

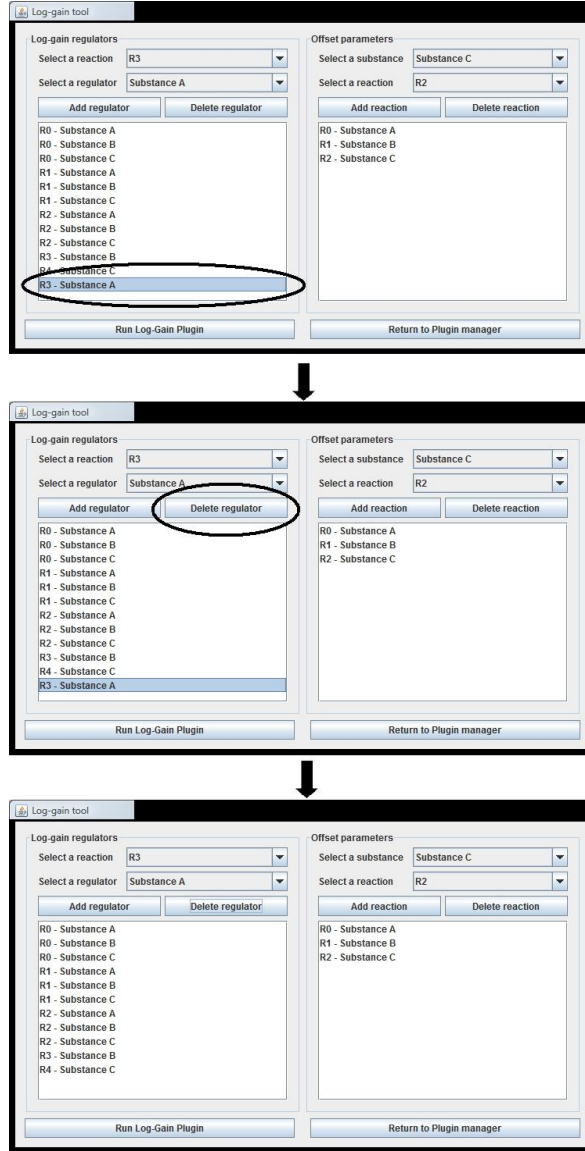


Figure 29: Deleting tuners: i) a reaction-tuner entry is selected; ii) the *Delete regulator* button is clicked; iii) the list contain the reaction-tuner entry is updated.

be written by the user as linear combination of monomials of substances and parameters raised to integer powers. For instance, given the MP system of Figure 9c, a parametric function for flux $F1$ could be:

$$Flux1(A, B, C) = a_0 + a_1AC + a_2B^2C^{-1}.$$

In this case the goal of linear regression plug-in is to find the coefficients

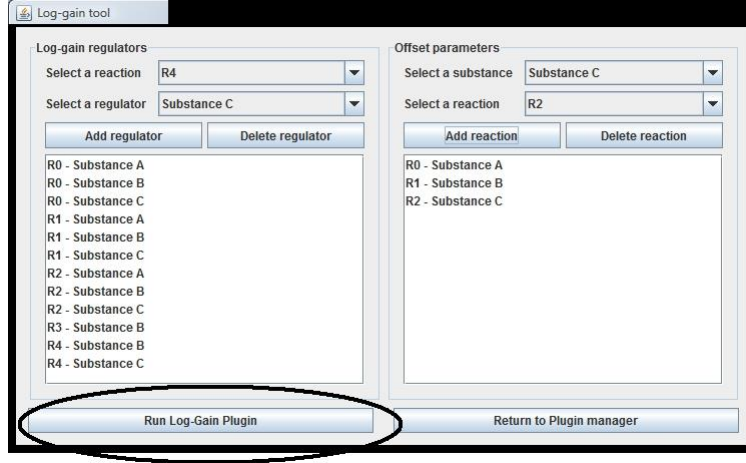


Figure 30: Start the flux discovery.

a_0 , a_1 and a_2 that make the regulation function of flux $F1$ fit a known time-series for this flux. In the following we introduce a few basic concepts related to linear regression models and then we show how they have been applied in our tool.

6.1 Short preliminaries about linear regression model

In regression analysis, often the variable of interest depends on more than just one other variable. Let us consider the general case it depends on a set of k independent variables, and let us describe the **k-multiple regression model**.

The regression model of a dependent variable Y on a set of k independent variables X_i , with $i = 1, \dots, n$, is given by the following equation:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_k X_k + \epsilon \quad (2)$$

where β_0 is said the *intercept* of the regression model, and each coefficient β_i , $i = 1, \dots, n$, represents the slope of the curve Y with respect to the variable X_i .

Notice that the independent variables of equation (2) may represent even non-linear functions.

If we consider a set of n numerical observations for each of the independent variables, then the equation (2) can be rewritten in the following manner:

$$Y_i = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \dots + \beta_k X_{ki} + \epsilon_i, \quad i = 1, 2, \dots, n \quad (3)$$

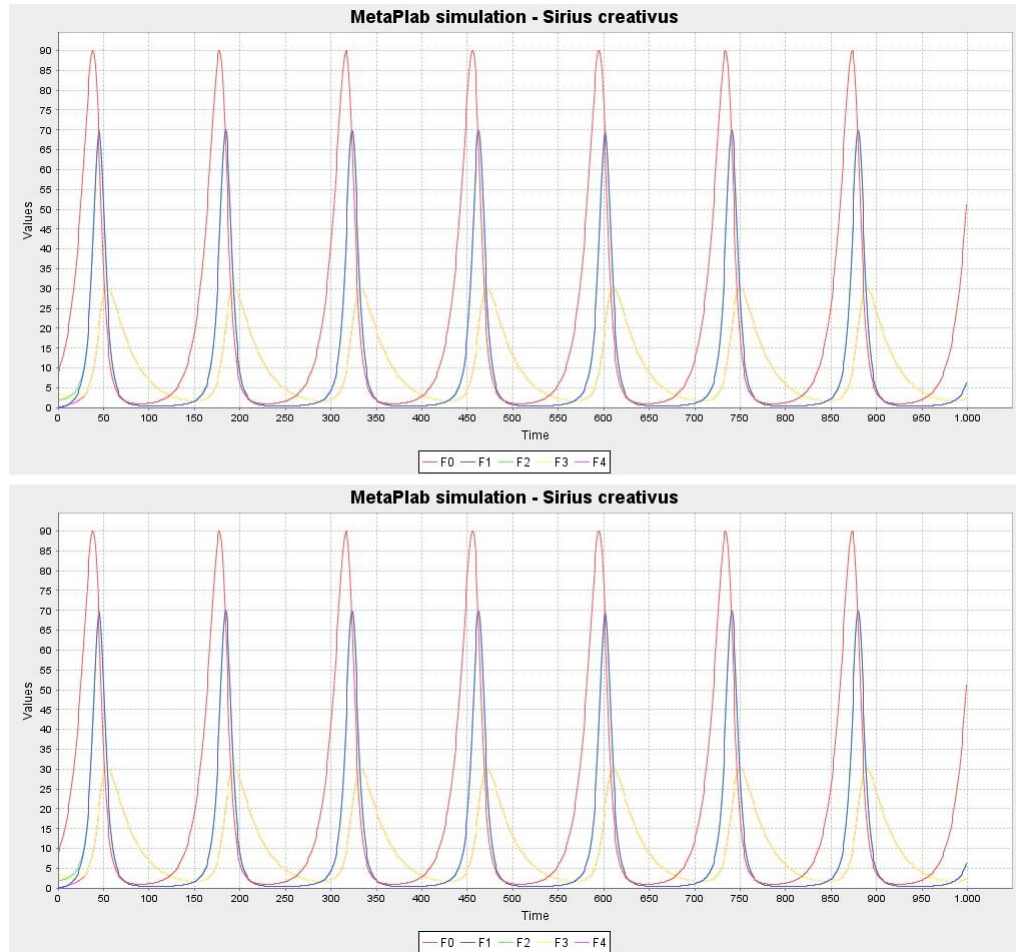


Figure 31: Fluxes obtained by Log-gain plugin (top) and by flux obtained by simulating Sirius model in Section 3 (bottom).

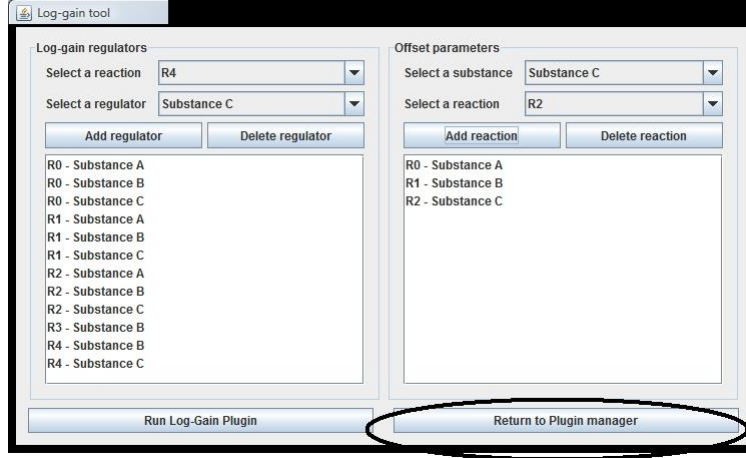


Figure 32: Returning to the plugin manager.

Then, equation (2) is given by

$$Y = X\beta + \epsilon \quad (4)$$

where $Y \in \mathbb{R}^{n \times 1}$, $X \in \mathbb{R}^{n \times k}$, β represents the parameters of the regression model and ϵ represents the errors, as in the following:

$$Y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix}, X = \begin{bmatrix} 1 & X_{11} & \cdots & X_{k1} \\ 1 & X_{12} & \cdots & X_{k2} \\ 1 & \vdots & \vdots & \vdots \\ 1 & X_{1n} & \cdots & X_{kn} \end{bmatrix}, \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}, \epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix} \quad (5)$$

A good estimation of the regression parameters β may be computed by means of the **method of least squares**.

We denote the least squares estimators of a regression parameter β_i by $\hat{\beta}_i$. The objective function S is defined as a sum of squared residuals r_j :

$$S = \sum_{j=1}^n r_j^2 \quad (6)$$

where each residual is the difference between the observed value and the value calculated by the model:

$$r_j = Y_j - \sum_{i=1}^k X_{ij} \hat{\beta}_i, \quad (7)$$

and the best fit is obtained when the sum of squared residuals is minimized.

From the theory of linear least squares, the parameter estimators are found by solving the normal equations, which can be written as:

$$(X^T X)\hat{\beta} = X^T Y \quad (8)$$

and then we compute the estimators of regression parameters along with the following formula:

$$\hat{\beta} = (X^T X)^{-1} X^T Y \quad (9)$$

6.2 Running linear regression plugin

In the following a comprehensive step-by-step procedure leads the user through the computation of MP regulation functions by the linear regression plugin.

Launching the plugin. In order to launch the linear regression tool we firstly launch MetaPlab by clicking on the MetaPlab.jar file or on one of the batch files (called MetaPlab-windows-xxxM or MetaPlab-linux-macOS-xxxM) that enable to allocate specific amounts of memory to the software. Afterwards, we load an MP model (.mps file) or we generate it from scratch, paying attention to enclose in the model file substance, parameter and flux time-series required to perform the linear regression. Time-series can be imported from external files or computed by *Dynamic computation* and *Log-gain* plug-ins (more details about these tools are available in the MetaPlab website). Once all these data have been collected, the linear regression plug-in can be launched by means of the *Plug-in manager* (highlighted in Figure 33). It displays the list of all the available plug-ins and enables the user to select and run them in a couple of clicks.

First interface: Polynomials generation. Linear regression plug-in employs two graphical user interfaces (GUIs). The first one appears as soon as the tool is launched and it enables the user to generate parametric polynomials. The second interface is displayed when the generation of polynomials is finished and it enables to compute the polynomial coefficients by linear regression. Let's we start explaining the first GUI, displayed in Figure 34.

By clicking on the topmost button we firstly select a workspace directory where the plug-in will automatically store files generated for the current MP model. Then, we select a reaction of the MP model, e.g., *R1* in Figure 34, and we start to generate one or more parametric polynomials which will be employed in the following regression stage. Each polynomial is a linear combination of substance and parameter variables representing, respectively,

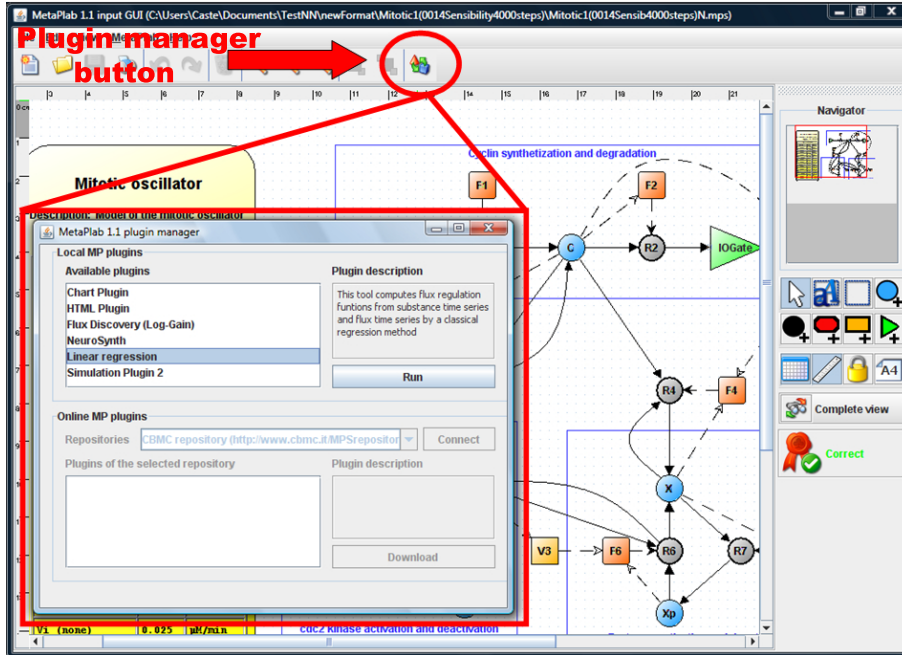


Figure 33: Calling the *Plugin manager* by clicking on the highlighted button in the MetaPlab toolbar. Available plug-ins are listed on the left side of the Plugin manager and they can be launched by the *Run* button

substance concentrations and parameter values of the biological system. To insert a new substance (parameter) in a polynomial we select the substance (parameter) variable from the list called “substances and parameters”, on the right side of Figure 34, we raise it to a proper power and, if necessary, we multiply or add it to other substance and parameter variables by clicking, respectively, on “*” and “+” buttons. While writing a polynomial its elements are visualized step-by-step in a text box (see polynomial $V1 + C * M$ in Figure 34). When all substance and parameter variables have been inserted we write a polynomial description in the bottom text box and we click on the “OK” button to add the new polynomial to the list of “polynomials available for reaction R_i ”, where R_i depends on the reaction selected before. Polynomials can be deleted from this list by clicking the *Remove* button. The overall process has to be repeated for each reaction of the inspected model, until each reaction is associated with at least one polynomial. At that point, we click on the *Next* button which opens the second GUI, described in the section below, and creates a text file for each reaction (e.g., R1.txt, R2.txt, etc.) and saves inside of them the polynomials generated so far.

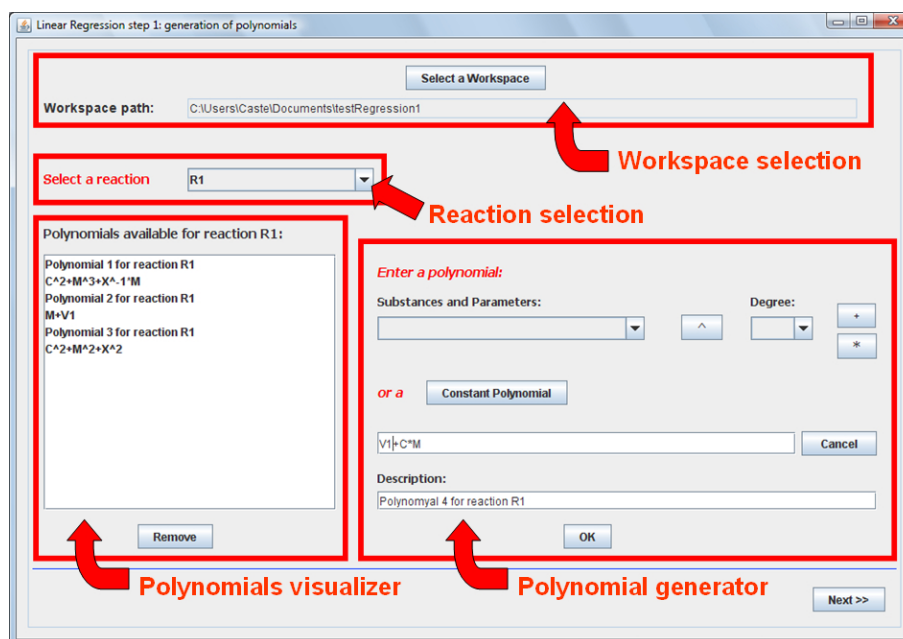


Figure 34: First GUI. Generation of parametric polynomials.

Second interface: Performing linear regression. The second GUI is composed of three main sections, as displayed in Figure 35. By the top-most section we can easily select one polynomial (among those generated by the first GUI) for each reaction. We firstly select a reaction from a list and then we use the *Add* and *Remove* buttons to select and deselect polynomials. Once a specific set of polynomials has been selected, we perform the linear regression process in order to compute coefficients that make polynomials fit the time-series data of substance, parameters and fluxes stored in the MP model. Results, i.e., optimized polynomials, are showed in the central part of the interface. The user can analyze these results and choose if to save them in a file as *regression experiments* with a suitable description or to discard them performing a new regression process. Saved experiments are stored in a text file called *exp.txt*, in the workspace, and they immediately appear in the third part (below) of the interface. On the left side of this section experiment descriptions are listed. The user can select them, visualize the related polynomials by clicking the “>>” button and send them to the MP model, one for each reaction, as regulation functions.

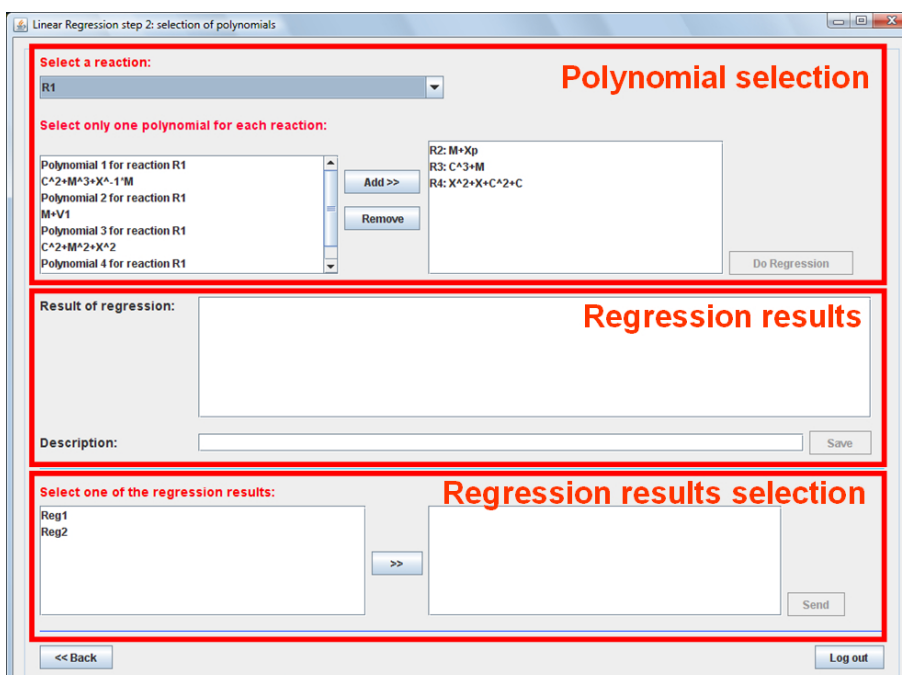


Figure 35: Second GUI. Linear regression

6.3 Managing linear regression workspace

Before executing the plugin, some notes have to be emphasized. This program is organized along two interfaces and the user can switch between them. In these interfaces there are some indirect operations of creating and/or deleting text files and creating folders, so the user has not to directly manipulate the workspace (folder) where he/she is working. Otherwise there could be a wrong execution of the plugin.

When the program has been launched, the user has to select a workspace. The folder selected can be there because referring to another graph, in this case a warning or an error message can appear. In the first case the user can still work in the folder, while in the second case the user must select another folder. Two cases can be found in the workspace:

- the number of the reactions of the loaded graph is lower than the text files present in the folder, and the file called *exp.txt* is not present;
- the number of the reactions is greater than the number of the text files with the name of the reactions, and there is a file named *exp.txt*.

In this second case, if the user is positive that the *exp.txt* file is relevant for his/her estimations, there are two possibilities to avoid execution errors:

1. manually create an empty text file with the name of the missing reaction(s);
2. cut and paste the *exp.txt* file outside the workspace; launch the application for adding new polynomials to the missing reaction(s); close the application; paste the previous *exp.txt* file in the current workspace and finally execute the plugin.

In order not to modify the workspace content, the user can open a new empty folder and work there (in this case the plugin creates all the files needed).

6.4 Generating Sirius' regulation functions

The plugin has been initially tested on *Sirius* model, introduced in Section 3. Other variants have been tested by introducing parameters or by modifying the time-series values or by changing the components name. Finally a check of correctness of the output computed by the tool has been performed by means of *Matlab*.

Given the following substances time-series:

$$A = \begin{pmatrix} 100.0 \\ 101.8676 \\ 103.8088 \\ 105.825 \\ 107.9178 \\ 110.0885 \\ 112.3387 \\ 114.6701 \\ 117.0843 \\ 119.5829 \end{pmatrix}; \quad B = \begin{pmatrix} 100.0 \\ 96.1727 \\ 92.5285 \\ 89.0596 \\ 85.7586 \\ 82.6185 \\ 79.6326 \\ 76.7944 \\ 74.098 \\ 71.5375 \end{pmatrix}; \quad C = \begin{pmatrix} 1.0 \\ 2.848 \\ 4.5872 \\ 6.2241 \\ 7.7646 \\ 9.2143 \\ 10.5786 \\ 11.8628 \\ 13.0716 \\ 14.2097 \end{pmatrix}$$

and the fluxes time-series below, which have been computed by the Log-gain plugin:

$$F0 = \begin{pmatrix} 3.7729 \\ 3.84481 \\ 3.91909 \\ 3.99613 \\ 4.0757 \\ 4.15816 \\ 4.24364 \\ 4.33207 \\ 4.4235 \end{pmatrix}; \quad F1 = \begin{pmatrix} 0.0189 \\ 0.05418 \\ 0.08829 \\ 0.12152 \\ 0.1540 \\ 0.18585 \\ 0.21717 \\ 0.24804 \\ 0.2785 \end{pmatrix}; \quad F2 = \begin{pmatrix} 1.8864 \\ 1.8484 \\ 1.8145 \\ 1.7818 \\ 1.7510 \\ 1.7221 \\ 1.69507 \\ 1.66983 \\ 1.64636 \end{pmatrix};$$

$$F3 = \begin{pmatrix} 3.8462 \\ 3.6983 \\ 3.55719 \\ 3.42252 \\ 3.2941 \\ 3.17175 \\ 3.0553 \\ 2.9444 \\ 2.83903 \end{pmatrix}; \quad F4 = \begin{pmatrix} 0.0385 \\ 0.11023 \\ 0.17769 \\ 0.24131 \\ 0.3013 \\ 0.35781 \\ 0.41087 \\ 0.46103 \\ 0.5086 \end{pmatrix}$$

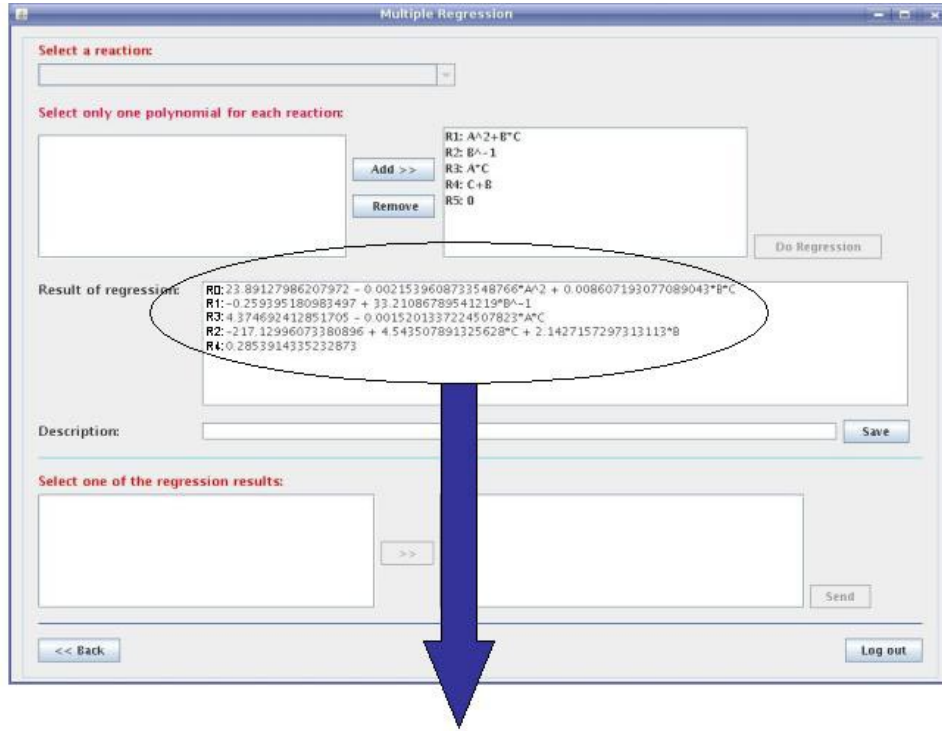
We load these time-series by the MetaPlab input GUI (see Section 3) and then we launch the linear regression plugin. The following polynomials have been generated through the first interface:

$$\begin{aligned} Poly0 &: A^2 + B * C, \\ Poly1 &: B^{-1}, \\ Poly2 &: C + B \\ Poly3 &: A * C \\ Poly4 &: 0 \end{aligned}$$

Afterward, we pass to the second interface, we select the polynomials just inserted and we perform linear regression which compute the following coefficients:

$$\begin{aligned} Poly0 &: 23.8912 - 0.0022 * A^2 + 0.0086 * B * C, \\ Poly1 &: -02594 + 33.2075 * B^{-1}, \\ Poly2 &: -223.8820 + 4.6858 * C + 2.2090 * B \\ Poly3 &: 4.3747 - 0.0014 * A * C \\ Poly4 &: 0.2607 \end{aligned}$$

In the picture below the *second* interface is depicted after the regression performance: the values obtained are identical to that achieved by performing linear regression by *Matlab*.



R0: 23.89127986207972 - 0.0021539608733548766*A^2 + 0.008607193077089043*B*C
 R1: -0.259395180983497 + 33.21086789541219*B^-1
 R3: 4.374692412851705 - 0.0015201337224507823*A*C
 R2: -217.12996073380896 + 4.543507891325628*C + 2.1427157297313113*B
 R4: 0.2853914335232873

Figure 8: Polynomials and coefficients generated by linear regression plugin.

7 Work in progress

The plugin framework explained at the beginning of Section 2 makes MetaPlab an extensible software. Whoever wants to process MP model data in a specific way can implement a new Java plugin by extending an abstract class contained in the MetaPlab source code. New plugins files inserted into the *pluginExt* folder of the MetaPlab package are automatically loaded when the software starts. Our research group is currently developing the following plugins:

- NeuroSynth: generates flux regulation functions and infers their tuners by means of a non-linear regression technique based on neural networks;

- Initial flux discovery: automatically computes initial flux values used by the Log-gain plugin to generate flux time-series;
- Covering discovery: automatically generates sets of reactions satisfying the Covering Offset Log Gain Property. These sets are used by the Log-gain plugin;
- Tuners discovering by statistical correlation: it looks for substances and parameters which can be employed as tuners for regulation functions;
- Regulation functions discovery by step-wise regression: employs the step-wise regression techniques in order to generate regulation functions;
- XML exportation: manages the translation of MP models to the XML standard.

References

- [1] L. Bianco and A. Castellini. Psim: a computational platform for Metabolic P systems. In *LNCS 4860*, pages 1–20. Springer, 2007.
- [2] L. Bianco, V. Manca, and F. Fontana. Reaction-driven membrane systems. In *Advances in Natural Computation, LNCS 3611*, pages 1155–1158. Springer, 2005.
- [3] L. Bianco, F. Fontana, G. Franco, and V. Manca. P systems for biological dynamics. In *[10]*, pages 81–126. Springer, 2006.
- [4] L. Bianco, F. Fontana, and V. Manca. P systems with reaction maps. *International Journal of Foundations of Computer Science*, 17(1):27–48, 2006.
- [5] L. Bianco, V. Manca, L. Marchetti, and M. Petterlini. Psim: a simulator for biochemical dynamics based on P systems. In *IEEE CEC2007 Volume*, pages 883–887, 2007.
- [6] A. Castellini and V. Manca. MetaPlab: A computational framework for metabolic P systems. In *LNCS*, volume 5391, pages 157–168. Springer-Verlag, 2009.
- [7] A. Castellini, V. Manca, and L. Marchetti. MP systems and Hybrid Petri Nets. In *Studies in Computational Intelligence*, volume 129, pages 53–62. Springer, 2008.

- [8] A. Castellini, G. Franco, and V. Manca. Hybrid functional Petri nets as MP systems. *Natural Computing*, 9121, 2009. DOI: 10.1007/s11047-009-9121-4.
- [9] A. Castellini, G. Franco, and V. Manca. Toward a representation of hybrid functional Petri nets by MP systems. In Y. Suzuki et al., editor, *Natural computing*, volume PICT 1, pages 28–37. Springer-Verlag, Tokyo, 2009.
- [10] G. Ciobanu, G. Păun, and M.J. Pérez-Jiménez, editors. *Applications of Membrane Computing*. Springer, 2006.
- [11] F. Fontana and V. Manca. Discrete solutions of differential equations by metabolic P systems. *Theoretical Computer Science*, 372:165–182, 2007.
- [12] F. Fontana and V. Manca. Predator-prey dynamics in P systems ruled by metabolic algorithm. *BioSystems*, 91(3):545–557, 2008.
- [13] F. Fontana, L. Bianco, and V. Manca. P systems and the modelling of biochemical oscillations. In *Membrane Computing, WMC 2005, LNCS 3850*, pages 199–208. Springer, 2005.
- [14] G. Franco and V. Manca. A membrane system for leukocyte selective recruitment. In A. Alhazov, C. Martín-Vide, and G. Păun, editors, *Membrane Computing, WMC 2003, LNCS 2933*, pages 180–189. Springer, 2004.
- [15] A. Goldbeter. A minimal cascade model for the mitotic oscillator involving cyclin and cdc2 kinase. *PNAS*, 88(20):9107–9111, 1991.
- [16] V. Manca. MP systems approaches to biochemical dynamics: Biological rhythms and oscillations. In *Membrane Computing, WMC 2006, LNCS 4361, 8699*. Springer, 2006.
- [17] V. Manca. Metabolic P systems for Biochemical Dynamics. *Progress in Natural Science*, 17(4):384–391, 2007.
- [18] V. Manca. Discrete Simulation of Biochemical Dynamics. In *DNA 13, LNCS 4848*, pages 231–235. Springer, 2008.
- [19] V. Manca. The metabolic algorithm for P systems: Principles and Applications. *Theoretical Computer Science*, 404:142–157, 2008.

- [20] V. Manca. Log-gain Principles for Metabolic P Systems. In *Natural Computing Series*. Springer, 2008. In print.
- [21] V. Manca. Fundamentals of metabolic P systems. In G. Păun, G. Rozenberg, and A. Salomaa, editors, *Handbook of Membrane Computing*, chapter 16. Oxford University Press, 2009. To appear.
- [22] V. Manca. Metabolic P dynamics. In G. Păun, G. Rozenberg, and A. Salomaa, editors, *Handbook of Membrane Computing*, chapter 17. Oxford University Press, 2009. To appear.
- [23] V. Manca and L. Bianco. Biological networks in metabolic P systems. *BioSystems*, 91(3):489–498, 2008.
- [24] V. Manca and L. Marchetti. XML Representation of Metabolic P systems. Accepted to the IEEE Congress on Evolutionary Computation (IEEE CEC 2009), Trondheim, Norway., 2009.
- [25] V. Manca, L. Bianco, and F. Fontana. Evolutions and oscillations of P systems: Application to biological phenomena. In *Membrane Computing, WMC 2004, LNCS 3365*, pages 63–84. Springer, 2005.
- [26] V. Manca, G. Franco, and G. Scollo. State transition dynamics: basic concepts and molecular computing perspectives. In M. Gheorghe, editor, *Molecular Computational Models: Unconventional Approachers*, chapter 2, pages 32–55. Idea Group Inc. UK, 2005.
- [27] V. Manca, R. Pagliarini, and S. Zorzan. A Photosynthetic process modeled by a metabolic P system. *Natural Computing*, DOI: 10.1007/S11047-008-9104-X, 2008.
- [28] E.O. Voit. *Computational Analysis of Biochemical Systems*. Cambridge University Press, 2000.